An Error Correction Neural Network for Stock Market Prediction

by

Mhlasakululeka Mvubu

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Mathematics in the Faculty of Science at Stellenbosch University

Supervisor: Prof. Jeff Senders

Co-supervisor: Prof. Ronnie Becker
Dr Bubacarr Bah (Thesis Advisor)

April 2019

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: April 2019

Abstract

An Error Correction Neural Network for Stock Market Prediction

Mhlasakululeka Mvubu

Department of Mathematical Sciences,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MSc

April 2019

Predicting stock market has long been an intriguing topic for research in different fields. Numerous techniques have been conducted to forecast stock market movement. This study begins with a review of the theoretical background of neural networks. Subsequently an Error Correction Neural Network (ECNN), Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) are defined and implemented for an empirical study. This research offers evidence on the predictive accuracy and profitability performance of returns of the proposed forecasting models on futures contracts of Hong Kong's Hang Seng futures, Japan's NIKKEI 225 futures, and the United State of America S&P 500 and DJIA futures from 2010 to 2016. Technical as well as fundamental data are used as input to the network. Results show that the ECNN model outperforms other proposed models in both predictive accuracy and profitability performance. These results indicate that ECNN shows promise as a reliable deep learning method to predict stock price.

Uittreksel

´n Fout Korrektiewe Neurale Netwerk vir voorspelling van aandeelmarkte.

("An Error Correction Neural Network for Stock Market Prediction ")

Mhlasakululeka Mvubu

Departement Wiskuudige Wetenskappe,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MSc

April 2019

Die voorspelling van die aandele mark was al lank ´n interge onderwerp in verskillende navorsingsvelde. Verskeie tegnieke was al so ver toegepas om aandelemark beweging te voorspel. Hierdie studie begin met ´n oorsig van die teoretiese agtergrond van neutrale netwerke. Daarna is ´n Fout Neurale Netwerk (FNN), Herhalende Neurale Netwerk (HNN); en Lank- en - Kort Termyn Gehee (LKTG) word gedefinieer en geïmplenteer vir ´n empiriese studie. Hierdie navorsing bied bewyse oor die voorspellende akkuraatheid en winsgewendheid van die opbrengste van die voorgestelde vooruitskatting modelle op termynkontrakte van; Hongkong se Hang Seng-toekoms, Japan se NIKKEI 225 termyne, en die Verenigde State van Amerika S&P 500 en DJIA termynkontrakte vanaf 2010 tot en met 2016. Resultate toon dat die FNN-model beter presteer as ander voorgestelde modelle in beide voorspellings akkuraatheid en winsgewendheid prestasie. Hierdie resultate dui daarop dat FNN belofte toon as ´n betroubaar masjienleermetode om die aandeelprys te voorspel.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor's Prof Ronnie Becker and Dr Bubacarr Bah for the continuous support of my MSc study and research, for their patience, motivation, enthusiasm, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor's and mentor for my MSc study.

A very special gratitude goes out to African Institute for Mathematical Science for providing the funding for me for all these years.

Finally, I must express my very profound gratitude to my parents Mzwakhe Justice Mvubu and Nomthunzi Mvubu and my brothers and sister for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Dedications

To my parents Nomthunzi Mvubu and late father Mzwakhe Justice Mvubu.

Contents

Declaration i

Abstract ii

Uittreksel iii

Acknowledgements iv

Dedications v

Contents vi

List of Figures viii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Problem Formulation . . . 2
- 1.3 Related Work . . . 2
- 1.4 Chapter Summary . . . 3

2 Background 5
- 2.1 Stock Market . . . 5
  - 2.1.1 Indices . . . 5
  - 2.1.2 Stock Market Data . . . 5
- 2.2 Neural Network . . . 7
- 2.3 The Single Neuron . . . 7
  - 2.3.1 Activation Functions . . . 7
- 2.4 Network Structure . . . 10
  - 2.4.1 Artificial Neural Network . . . 10
  - 2.4.2 Loss Function . . . 11
  - 2.4.3 Stochastic Gradient Descent . . . 12
  - 2.4.4 Backpropagation Algorithm . . . 13
- 2.5 History . . . 16

3 Recurrent Neural Networks 18
- 3.1 Introduction . . . 18
- 3.2 Unfolding Computation Graph for RNN . . . 18
  - 3.2.1 Overshooting in Recurrent Neural Network . . . 21
- 3.3 Backpropagation Through Time . . . 22
- 3.4 Vanishing and Exploding Gradient Problem . . . 25

3.5 Gradient-based Learning Methods . . . 27
3.5.1 Stochastic Gradient Descent . . . 27
3.5.2 Adaptive Gradient Algorithm (Adagrad) . . . 28
3.6 Long Short-Term Memory Neural Network . . . 28
3.6.1 Conventional LSTM . . . 29

4 Error Correction Neutral Network **32**
4.1 Overview of An Error Correction Neural Network . . . 33
4.2 Unfolding in Time of An Error Correction Neural Network . . . 34
4.3 Overshooting in Error Correction Neural Network . . . 35
4.4 Computing the Gradient in Error Correction Neural Network . . . 35
4.5 Extension of Error Correction Neural Network . . . 38
4.5.1 Variants-Invariants Separation . . . 38
4.5.2 State Space Reconstruction . . . 39
4.5.3 Undershooting . . . 39

5 Empirical Study **41**
5.1 Stock Data . . . 41
5.1.1 Data Section Descriptions . . . 42
5.1.2 Model Input . . . 43
5.1.3 Problem Definition . . . 45
5.2 Experimental Design . . . 47
5.2.1 Prediction Approach . . . 47
5.2.2 Training and Performance Evaluation . . . 47
5.2.3 Performance Criteria . . . 48
5.2.4 Trading Strategy . . . 49

6 Results **51**
6.1 Predictive Accuracy Test . . . 60
6.1.1 Directional Accuracy . . . 66
6.2 Profitability Test . . . 67

7 Conclusion and Future Work **70**
7.1 Conclusion . . . 70
7.2 Future Work . . . 71
7.2.1 Input Selection . . . 71
7.2.2 Sentiment Analysis . . . 71

List of References **72**

List of Figures

2.1 Two industrial revolutions showing jump discontinuities caused by stock split. Shiller PE Ratio (price/income) [61]. . . . 6
2.2 Biological neuron with axon, dendrites and cell body. The illustration is a subtle adaption [39]. . . . 8
2.3 An artificial neuron with threshold function, $\psi(y)$. . . . 8
2.4 Sigmoid function . . . 9
2.5 Tanh function . . . 9
2.6 ReLU function . . . 9
2.7 Softmax function . . . 9
2.8 Most common activation function . . . 9
2.9 A network with four layers. The edge corresponding to the weight w_{43}^3 is highlighted. The output from the neuron number 3 at layer 2 is weighted by factor w_{43}^3 when it is fed into neuron number 4 at layer 3. . . . 10

3.1 The classical dynamical system is described by Equation (3.2.3), illustrated as an unfolded computational graph. Each node represents the state at some time t and the function f maps the state at t to state at $t+1$. The same parameters (the same value of θ) are used for all time steps. . . . 19
3.2 The computational graph to compute the training loss of the recurrent network that maps an input into a sequence of x values to a corresponding sequence of output y values. A loss function L measures how far each y is from the corresponding training target $\hat{y}$ (left). The RNN and its loss draw with recurrent connections (right). 20
3.3 RNN incorporating overshooting . . . 21
3.4 As the network receives new input over time, the sensitivity of units decay (lighter grey shades in layer) and the back-propagation through time (BPTT) overwrites the activation function in the hidden units. . . . 24
3.5 gradient explosion clipping visualization, adopted from [51]. . . . 27
3.6 The detailed internals of an LSTM. . . . 29

4.1 Unfolded error correction neural network where the weight matrices A, B, C, and D are shared weights and $-Id$ is the negative identity matrix. An ECNN whose only recurrence is the feedback connection from the error $z^{(t-2)}$ calculated from previous time step $t-2$ to output to the hidden layer. At each time step t, the input is $x^{(t)}$, the hidden layer activations are $s^{(t+1)}$, the outputs are $y^{(t+1)}$. . . . 34
4.2 ECNN incorporating overshooting. Note that $-Id$ is the fixed negative of an identity matrix, while $z^{(t-1)}$ are outputs clusters to model the error correction mechanism (see comment in Figure 4.1). . . . 35
4.3 ECNN with Variant-Invariant Separation (see comments in Figure 4.1). . . . 39

4.4 The time series of observed state description $x^{(t)}$ may follow a very complex trajectory. A transformation to a possible higher-dimensional state space may result in a smother trajectory, adopted from [68]. 40

5.1 GSPC daily closing price development between the period of 2002-01-01 and 2015-12-30. 42
5.2 HSI daily closing price development between the period of 2002-01-01 and 2015-12-30 43
5.3 DJIA daily closing price development between the period of 2002-01-01 to 2015-12-30. 43
5.4 Nikkei 225 daily closing price development between the period of 2002-01-01 to 2015-12-30. 44
5.5 The is the data representational structure which has the shape (number of samples, rolling window, number of features) 46
5.6 Stock i from one set of training data from 1^{th} January 2002 to 10^{th} 46
5.7 Continuous data set arrangements during the entire sample period for training, validation and testing. 48

6.1 Comparisons of the predictive data and actual data for the forecasting models. η is the learning rate and rw rolling window. The different hyper-parameters give the best performance of the respective' models. 52
6.2 Comparisons of actual data and predicted HSI data using RNN, LSTM and ECNN with a window of 60 rolling window. 52
6.3 $\Big($(6.3a), (6.3b), (6.3c), (6.3e), (6.3f)$\Big)$ forecasting loss from the models. 53
6.4 Comparisons of the predictive data and actual data for the forecasting models. η is the learning rate and rw rolling window. The different hyper-parameters give the best performance of the respective' models. 54
6.5 Comparisons of actual data and predicted N225 data using RNN, LSTM and ECNN with a window of 60 rolling window. 54
6.6 $\Big($(6.6a), (6.6b), (6.6c), (6.6e), (6.6f)$\Big)$ forecasting loss from the models. 55
6.7 Comparisons of the predictive data and actual data for the forecasting models. η is the learning rate and rw rolling window. The different hyper-parameters give the best performance of the respective' models. 56
6.8 Comparisons of actual data and predicted DJIA data using RNN, LSTM and ECNN with a window of 60 rolling window. 56
6.9 $\Big($(6.9a), (6.9b), (6.9c), (6.9e), (6.9f)$\Big)$ forecasting loss from the models. 57
6.10 Comparisons of the predictive data and actual data for the forecasting models. η is the learning rate and rw rolling window. The different hyper-parameters give the best performance of the respective' models. 58
6.11 Comparisons of actual data and predicted S&P 500 data using RNN, LSTM and ECNN with a window of 60 rolling window. 58
6.12 $\Big($(6.12a),(6.12b),(6.12c),(6.12e),(6.12f)$\Big)$ forecasting loss from the models. 59
6.13 Comparisons of the predictive directional accuracy a forecasting models in different stock indices. 67

Chapter 1

Introduction

This chapter provides a context for the study. The motivation behind the research is given in Section 1.1 before the problem itself is described in Section 1.2. Section 1.3 presents similar or related research. Finally, the general structure of the thesis is outlined in Section 1.4.

1.1 Motivation

Financial time series forecasting is currently an important topic for many financial analysts and researchers, as precise forecasting of different financial applications plays a consequential role in decision-making on investment. Patterns found in financial time series data structure are then exploited to make predictions about the future which can be used to guide decision making and yield a significant profit. The forecasting of stock markets is one of the most difficult tasks for time-series analysis, as financial markets are influenced by numerous social, psychological and economic external factors that lead to irrational and unpredictable behavioral characteristics of stock prices. *Efficient Market Hypothesis* (EMH) is a theory in financial economics which states that asset prices fully reflect all the information available [64].

Forecasting financial data is subject to large error since financial time series are generally non-stationary, complicated and non-linear. However, it is possible to design mechanisms for prediction of financial markets [1], and developing more realistic models for predicting financial time series data to extract meaningful statistics of greater efficacy and accuracy is of great interest. Financial data mining and technical analysis with statistical and machine learning techniques have been used in this area to develop strategies and methods that could be useful for forecasting financial time series. The existing methods for stock price forecasting can be classified as follows,

- Fundamental Analysis
- Technical Analysis
- Time Series Forecasting

In [37] fundamental analysis is defined as a method of evaluating a security in an attempt to assess its intrinsic value, by examining related economic, financial, and other qualitative and quantitative factors. This method is most suited for long-term forecasting. Technical analysis uses the historical price data for identifying the future price. A typical statistic used for technical analysis is the moving average, which is the unweighted mean of a certain number of past data points. Time Series Forecasting is the use of a model to predict future

values based on previously observed values. Predictions can be made either by using linear or non-linear models. The traditional statistical linear models are autoregressive conditional heteroskedastic (ARCH) [20], generalized autoregressive conditional heteroskedastic (GARCH) [19], autoregressive integrated moving averages (ARIMA) [14], and Smooth Transition Autoregressive (STAR) models [58]. The main disadvantage of these models is that they fail to capture the complexity and latent dynamics of the stock data, as opposed to neural network models. If the system being studied is non- stationary and dynamic in nature, the neural network can in real time change its network parameters (synaptic weights). So, neural network suits better than other models in predicting the stock market price [46].

In the past decade, complex machine learning techniques often referred to as Deep Learning techniques have been used for time series prediction. Deep neural networks can be considered as non-linear function approximators. Various types of deep neural network architectures are used for different tasks. Among them are the Multi-Layer Perceptrons (MLP), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Convolutional Neural Networks (CNN) [9]. They have been applied in various areas such as image recognition, speech recognition, time series analysis etc., [26, 42]. Deep learning algorithms are able to reveal some insight about the underlying hidden patterns and underlying dynamics in the time series data through a self-learning process.

The focal point of this work is on the Error Correction Neural Network (ECNN) developed by Zimmermann et al.[71] and compares with RNN and LSTM models. The ECNN integrates prior knowledge of errors in the neural network model and increases its resilience to the overfitting problem [71]. Prior knowledge constitutes the view of stock markets as dynamic systems that transform the forecasting problem into a system identification task. The core objective is to develop a suitable dynamical system that clearly explains the underlying patterns for stock market predictions. The error obtained from the previous prediction is utilized as an additional input force in the ECNN in order to guide the model dynamics. Grothmann [70] found that the an error correction neural networks is a promising emerging solution to the financial market forecasting problem.

1.2 Problem Formulation

In this study, a selection of deep neural network models is applied to short-term forecasting. Models are trained independently using four selected indices as data to test the prediction ability of each one and the results are compared to M'ng and Mehralizadeh [49] and Wei et al.[6]. In addition to the prediction ability, a six-yearly profitable test of returns is performed in each model. The selected indices are Nikkei 225, Standard and Poor's 500 (S&P 500), Dow Jones Industrial Average (DJIA), Hang Seng Index (HSI).

1.3 Related Work

In the context of predicting time series neural networks, there is much research available. M'ng and Mehralizadeh [49] and Wei et al.[6] present a new deep learning framework where wavelet transforms (WT), stacked autoencoders (SAEs) and long short-term memory (LSTM) are combined for stock price forecasting. Their framework develop a six years predictive performance of the four proposed models in different stock markets (see [6]), WT-LSTM, SAEs-LSTM, RNN, and LSTM. Where SAEs-LSTM is the main part of their model and is used to learn the deep features of financial time series.

1.4 Chapter Summary

Following this introduction are six chapters: Background, Recurrent Neural Networks, Error Correction Neural Networks, Empirical Study, Results, Conclusion and Future Work.

Chapter 2: Background

The background chapter covers relevant techniques and concepts that this work revolves around. It also includes an introduction to stock markets, a brief introduction to the building blocks of artificial neural networks in its broader scope and networks specializing in forecasting the movement of financial markets. It provides information on certain concepts required for the following chapters, covering recurrent neural networks, long-term memory and error correction neural networks in relation to the forecasting problem.

The information presented can be divided into two parts relating to an artificial neural networks. The first part deals with basic information on the foundations of artificial neurons are building blocks of neural networks. A feed-forward neural network is also discussed in Chapter 2. In addition to the general information, this chapter also discusses the learning process and covers general learning rules and techniques in addition to the general information. Finally, a brief history of early neural network designs and different areas of applications are also provided.

Chapter 3: Recurrent Neural Network

This chapter presents a brief introduction to recurrent neural networks. Recurring neural networks concept can be broken down into four parts. The first part covers the unfolding computation graph of a recurrent neural network. The second part, the overshooting in recurrent neural networks. The third and fourth part focuses on the learning algorithm, often referred to as back-propagation-through time. A brief description of issues related to exploding and vanishing gradients are discussed.

The last part focuses on a general overview of the long short-term memory technique which is an extension of recurrent neural networks. A brief summary of a conventional LSTM is provided.

Chapter 4: Error Correction Neural Network

The main point of focus in this thesis is on error correction neural network, which is broadly described in this chapter. Some important concepts like finite unfolding in time and overshooting are provided. For an empirical study, an error correction neural network is built and implemented. It will be then concluded with a description of some of the provided extensions, applicable for error correction neural networks.

Chapter 5: Empirical Study

In this chapter, an empirical study is carried out to investigate an optimal configuration, training procedures and the impact of the technical indicators on model performance for the proposed models. The selection of data and how data are preprocessed are also discussed. The selected stock market indices are briefly described in this chapter.

In addition to this brief description of prediction approach, training and performance evaluation of the models applied in this study is given. Finally, the candidate models are presented along with their respective hyperparameters.

Chapter 6: Results

Results obtained for each model are presented and discussed in Chapter 6. Section 6.1 considers each model separately, stating its predictive performance and accuracy measure as well as comparing it to the M'ng and Mehralizadeh [49] and Wei et al.[6] results. Lastly, Section 6.1.1 and 6.2 analyze the directional accuracy and profitability of each model.

Chapter 7: Conclusion and Future Work

Chapter 7 concludes the thesis, emphasizing notable findings in light of the introductory problem statement. Section 7.2 highlights relevant topics that could be considered for further research.

Chapter 2

Background

This chapter describes models and research relevant to the study in this thesis. Section 2.1 provides introductory material to stock markets, while Section 2.2 describes what neural networks are and how they work.

2.1 Stock Market

Prediction of the stock market has long been an intriguing topic for researchers in different fields. This is commonly viewed as a very difficult task, partly contributed by the random walk behavior of the stock market movements. [30]. In addition, financial time series (e.g. stock markets) have inherent characteristics such as non-linearity, outliers, missing values, complex and chaotic nature of the system [52].

A *Stock exchange* is an institution, organization, or association which hosts a market where the stocks, bonds, options and futures, and commodities are traded. A stock which consists of high trading volumes, whose shares are readily available for trading, is often referred to as a *Liquid Stock*. The liquidity of a single share is based on its trade on the stock exchange, reflecting the investor's demand for a fixed supply of shares. The more activities in the market, the easier it is to find a buyer when someone tries to sell and vice versa. For less liquid stocks some participants might struggle to complete trades.

2.1.1 Indices

An *index* is a collection of stocks, whose values are based on the underlying share prices. It is computed from the prices of selected stocks (typically a weighted average). Since stock indices represent the aggregation of multiple stocks, they are often consulted to sample the stock market as a whole. In this paper, we have used specific major stock indices (see Section 6.3).

2.1.2 Stock Market Data

Various types of data are available for a stock, ranging from fine-grained information concerning each trade to one *data point* every month [30]. Exactly what the data points contain may vary, but a widely used composition consists of six variables: *Time*, *open*, *high*, *low*, *close* and *volume*. *Time* is a reference to when the data point is from. The share price traded at the beginning and at the end of the regular trading period is *open* and *close*, whereas *high* and *low* refer to the highest and lowest point the value achieved during the trading period.

Lastly, *volume* is the number of shares that were traded in total during a regular trading period. Time series data of stock prices are non-stationary and undesirable to utilize in their raw form when forecasting [30]. According to New York Stock Exchange (NYSE), there are approximately 252 trading days a year [59].

Stock Split

A *stock split* is a corporate action where the company divides the existing outstanding shares in order to boost the liquidity of shares. For example, if a company publicly announces its earnings after the exchange has closed, the market is likely to *gap* (the difference between the supply and demand for that product) the next workday. Large jump discontinuities (see Figure 2.1) are observed when a company decides to *split* its shares, the value of each share is halved and the number of shares is doubled. Stock splitting is not a frequent phenomenon and stock data are easily adjusted to remove the gaps.

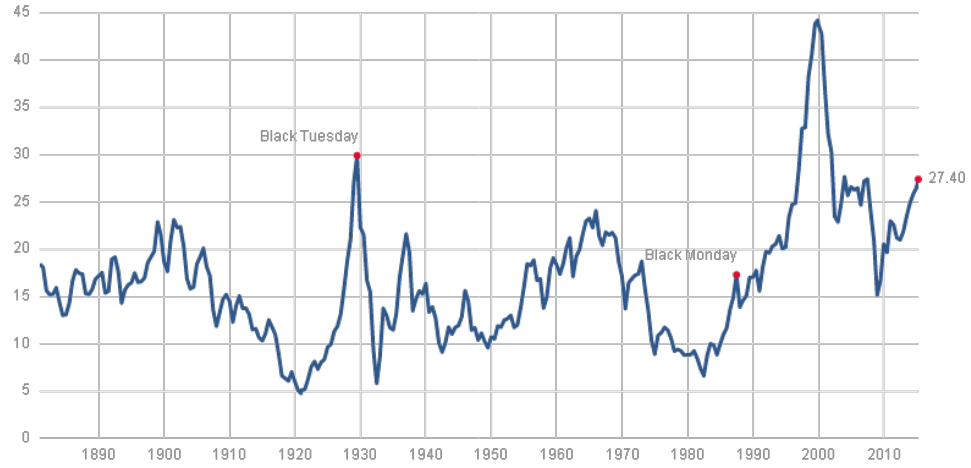


Figure 2.1: Two industrial revolutions showing jump discontinuities caused by stock split. Shiller PE Ratio (price/income) [61].

Changing Trends

Observing the historical distribution of market price may provide clues to the development of future price. Factors ranging from macroeconomic changes and natural disaster to manage illness and product releases contribute to the price changes over the short and long term and can provide insight into how future trends may occur. In addition, we also have prominent major factors like government, international transactions, speculation and expectation which contribute hugely to changes in stock price. These important factors may be different from each other, they are interconnected. Current affairs and Government mandates can influence international transactions that have a role in speculation. Shifts in supply and demand can impact each of these factors. This is all based on personal perception. Everything is based on personal perception. If people think that a company will do better in the future, the demand and price of the stock will increase, and if they think that a company will do worse, the demand and price of the stock will be lowered [2]. Investors often refer to markets that are expected or are increasing as *bull* markets, caused by more people want to buy their stocks than to sell them. Conversely, *bear* markets are declining markets caused by more people want to sell a stock than buying it.

2.2 Neural Network

The foundation of neural networks in a scientific sense begins with biology. A biologist Emerson M. Pugh once said, "If the human brain were so simple that we could understand it, we would be so simple that we couldn't" [38]. The conception of how the brain works remains relatively unknown, but is that hasn't deteriorated the efforts of those in the field of neuroscience, to unfold and understand brain functioning. The fundamental description of the human brain can be described as the composition of several neural cells (Biological neurons), each consisting of an estimated 10 billion neurons (nerve cells) and 6000 times as many synapses (connections) between them [29]. In order to build artificial neural networks(ANN), computer scientists have adapted this understanding of the brain. A general overview of ANNs is given in Section 2.4.1.

2.3 The Single Neuron

We begin with a fundamental description of the human brain. The human brain is composed of several neural cells (biological neuron), each consisting of a cell body, dendrites, and an axon (see Figure 2.2) [7]. In the human brain, a typical neuron receives a signal through the axon, it either inhibits or excites this signal and passes it on through its dendrites to all the connected neurons [7]. The artificial neuron is best illustrated by analogy with the biological neuron. Figure 2.3 conceptually depicts an artificial neuron.

Similar to biological neural networks, the ANN consist of connected artificial *neurons*, also known as *units* or *nodes*, every node has certain number of input and output channels. The connections w_i transfer the signal x_i from one neuron to another through dendrites (see Figure 2.2, 2.3). A weight matrix w_i represent the "importance" of that specific input x_i. The artificial neuron then multiplies each of these inputs by a weight, then it adds the multiplications and passes the sum to an activation function. If the sum exceeds an external threshold θ, the neuron emits output z. In this case, z is either continuous or binary value depending on the activation function. Depending on the activation function, z is either a continuous or a binary value. In most cases, the activation function that converts the neuron output to $[0,1]$ or $[-1,1]$ intervals.

$$y = \sum_{i=1}^{n} w_i x_i - \theta \tag{2.3.1}$$

and

$$z = \psi\Big(y\Big) \tag{2.3.2}$$

where y is the net input, θ the threshold and $\psi(.)$ the activation function. Figure 2.3 conceptually illustrates an artificial neuron. The following Section 2.3.1 briefly summarises different activation functions and their appropriate selection depending on the nature of the problem.

2.3.1 Activation Functions

There are multiple activation functions to choose from. This section summarizes briefly some of the most popular activation functions. In neural computing, four different types of activation functions are used almost exclusively. They are presented by Figure 2.4, 2.5, 2.6, 2.7. The " sigmoid," " tanh," " softmax" and " rectified linear unit (ReLU) have recently received

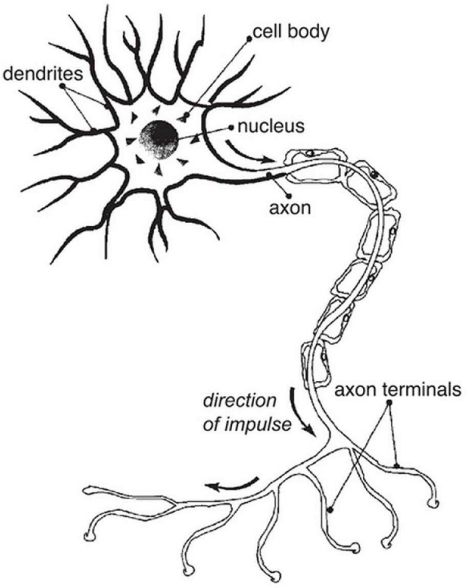


Figure 2.2: Biological neuron with axon, dendrites and cell body. The illustration is a subtle adaption [39].

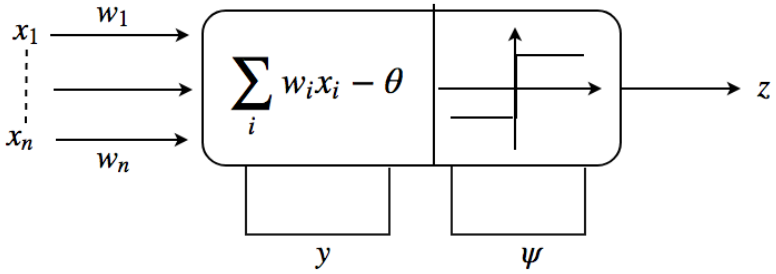


Figure 2.3: An artificial neuron with threshold function, $\psi(y)$.

more attention than the other activation functions (e.g. Gaussian, Sinc, etc.). The "tanh" and "sigmoid" activation functions are defined as follows:

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.3.3}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3.4}$$

respectively. The "sigmoid" defined by Equation (2.3.4) is a common choice that takes real values into the range $[1, 0]$, this is normally used in the output layer. The "tanh" activation function is an elegant way to "squash" real values into the $[-1, 1]$ range, preserving the sign and conforming to the boundary condition $f(0) = f'(\pm\infty) = 0$. ReLU is another popular activation function for positive input values that is open-ended [10], defined as

$$y(x) = \max\Big(x, 0\Big) \tag{2.3.5}$$

Softmax is a generalization of logistic regression that takes a k-dimensional vector of arbitrary real values and produces another k-dimensional vector with real values in the range [0, 1] that

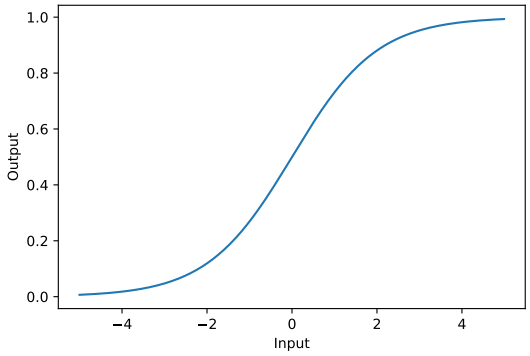


Figure 2.4: Sigmoid function

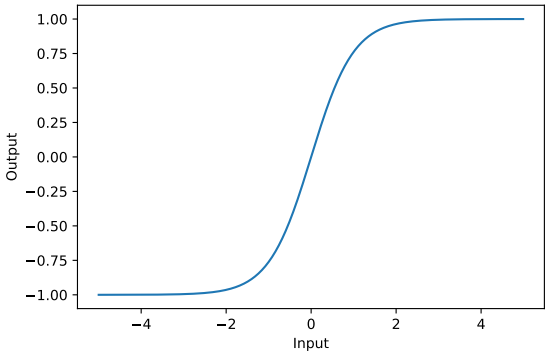


Figure 2.5: Tanh function

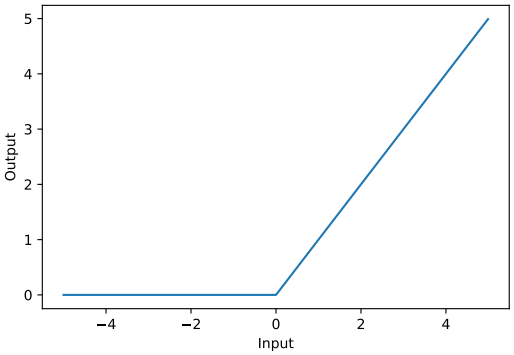


Figure 2.6: ReLU function

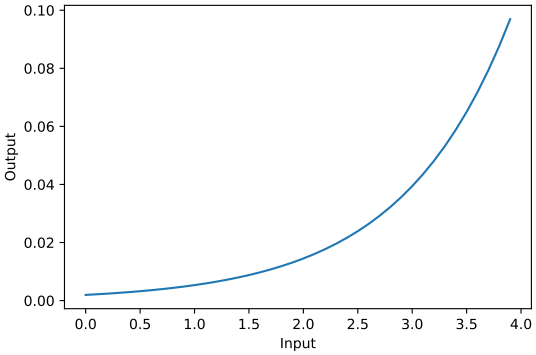


Figure 2.7: Softmax function

Figure 2.8: Most common activation function

add up to 1.0, and is defined by the following equation:

$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.3.6}$$

The purpose of a Deep Learning activation function is to ensure that the representation in the input space is mapped to a different output space. Selecting an appropriate activation function depends on the problem and the nature of the data. Threshold functions that return zero or one, based on whether the input is less or greater than a certain limit, were used as activation functions early on. Also often referred to as *perceptrons*, these networks are first introduced in [55]. On the other hand, the "sigmoid" function is suitable for network models where one has to predict probability as an output within the range $[0, 1]$ (see Figure 2.4). The ReLU activation function shown in Figure 2.6 speeds up the convergence of stochastic gradient (SGD) (see Section 2.4.3). It is argued that this is due to its linear, non-saturating form [40]. ReLU is computationally cheap since it can be implemented by thresholding an activation value at zero. However, ReLU units can be fragile during training and can "die". For example, a large gradient flowing through a ReLU neuron could cause weights to update in such a way that the neurons will never activate on any data-point again. If the task in a network model is a classification problem, the use of the computationally expensive softmax (shown by Figure 2.7) function for the output layer may be considered. This function ensures that all outputs sum to one, resulting in values that resemble probabilities.

2.4 Network Structure

Powerful machine learning models such as Deep Neural Networks (DNN) have been successful in different artificial intelligence tasks. Although different architectures and modules have been proposed for DNNs, it is a challenge to select and design the appropriate network structure for a target problem.

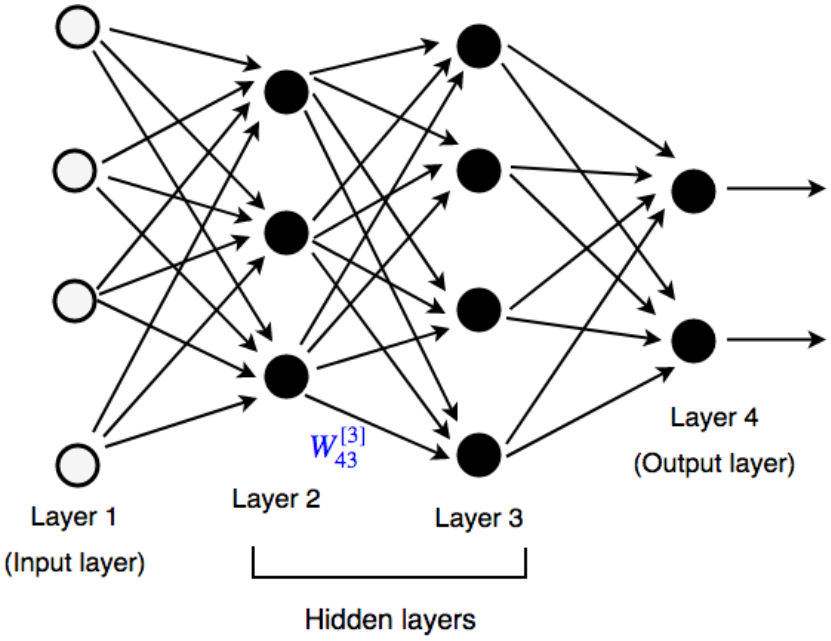


Figure 2.9: A network with four layers. The edge corresponding to the weight w_{43}^3 is highlighted. The output from the neuron number 3 at layer 2 is weighted by factor w_{43}^3 when it is fed into neuron number 4 at layer 3.

2.4.1 Artificial Neural Network

An artificial neural network is defined as an electronic model based on the neural structure of the brain which was first developed by McCulloch and Pitts published in 1943 [43]. The article showed that even simple types of neural networks could compute any arithmetic or logistic function. This article was widely read and had great influence in many academic fields. This first model was known as *Neurocomputing*. The model had two inputs and a single output. The McCulloch and Pitts' neuron has become known today as the *logic circuit* [22].

Figure 2.9 represents the architecture of a simple neural network. It is made up from an input, output and one or more hidden layers. Each node from input layer is connected to a node from hidden layer and every node from the hidden layer is connected to every node in the next hidden layer, or if it is the last hidden layer, to every node of the output layer. There is usually some weight associated with every connection. At the input layer, there is no "previous layer" and each neuron receives the input vector that is fed into the network. At the output layer, there is no "next layer" and these neurons provide the overall output.

The diagram shown in Figure 2.2 and 2.3 is a biological structure of a neuron which relates to an artificial neural network with some brief basic mathematical modeling. The ANN ar-

chitecture shown in Figure 2.9 is explained in detail as follow: the Multi-Layered Perceptron (MLP) is arranged in four layers, with the first layer taking inputs, and last layer producing output. The middle layers are known as hidden layers since they are not connected with the external world. The first layer weighs the input evidence then transmits to the first hidden layer, then to the second hidden layer. This type of layer makes a decision at a more complex and abstract level than the first layer. And the fourth layer produces an output and returns a neural network.

We suppose that the network has L layers, where layers 1 and L are the input and output layers, respectively. Suppose that layer l, for $l = 1, 2, 3, ..., L$ contains n_l neurons. So n_1 is the dimension of the input data. Overall, the network maps from $\mathbb{R}^{n_1}$ to $\mathbb{R}^{n_L}$. We use $W^l \in \mathbb{R}^{n_l \times n_{l-1}}$ to denote the matrix of weights at layer l. More precisely, to rewrite this expression in a matrix form we define a *weight matrix* W^l for each layer, l. Each node at position k of layer $l-1$ is joined to each node at position j of layer l. by an edge of weight w^l_{jk}. Similarly, $b^l \in \mathbb{R}^{n_l}$ is the vector of biases for layer l, so neurons j at layer l uses the bias b^l_j. In Figure 2.9 we give an example with $L = 4$ layers. Here, $n_1 = 4$. $n_2 = 3$, $n_3 = 4$ and $n_4 = 2$, so $W^2 \in \mathbb{R}^{3\times4}$, $W^3 \in \mathbb{R}^{4\times3}$, $W^4 \in \mathbb{R}^{2\times4}$, $b^2 \in \mathbb{R}^3$, $b^3 \in \mathbb{R}^4$ and $b^4 \in \mathbb{R}^2$.

Given the input $x \in \mathbb{R}^{n_1}$, we may then neatly summarize the action of the network by letting o^l_j denote the output, from neuron j at layer l. So,

$$o^1 = x \in \mathbb{R}^{n_1} \tag{2.4.1}$$

where o^1 is called the *input layer* and o^L is called the *output layer*. We also denote the values of o^1 by vector x of the size n_1. We will denote values of o^L by vector y of size n_L

$$o^l = \sigma\left(W^l o^{l-1} + b^l\right) \in \mathbb{R}^l, \quad \text{for } l = 2, 3, ..., L. \tag{2.4.2}$$

where $\sigma(.)$ is the sigmoid activation function described by Equation (2.3.4). Note that, Equation 2.4.1 and 2.4.2 amount to an algorithm for feeding the input forward through the network which results to an output $o^L \in \mathbb{R}^{n-L}$. Given a set of an input vector and a set of corresponding output vectors, we would like to choose parameters W^l, b^l such that our function maps each input into its corresponding output. In general, we only know the corresponding values of a proper subset of the inputs. We would then use the knowledge of the known correspondence to train the network (that is, identifying the parameters), which will have a high percentage success in identifying the corresponding vectors of the entire set. We will do a training process called *Backpropagation* (BP) which is described in Section 2.4.4. In the following sections, we describe a brief description of the following concepts: Loss Function, Stochastic Gradient Descent, Backpropagation Algorithm.

2.4.2 Loss Function

The main objective of the ANN in accordance with the architecture described in Figure 2.9 is to find a set of weights w and biases b that will minimize the loss function. Now suppose we have N pieces of data or training set, in $\mathbb{R}^{n_l}$, $\left\{x^{(i)}\right\}_{i=1}^N$, for which they are given the target outputs $\left\{y(x^{(i)})\right\}_{i=1}^N$ in $\mathbb{R}^{n_L}$.
In this case the quadratic function that we wish to minimize has the form:

$$E_{x^{(i)}}(W, b) = \frac{1}{2N} \sum_{i=1}^{N} || \, y(x^{(i)}) - o^L(x^{(i)}) \, ||^2, \tag{2.4.3}$$

The error is minimized through updates on the weights. This is explained in detail in Section 2.4.4. To minimize the loss function E, each weight w^l_{jk} is updated by an amount proportional to the partial derivatives of E with respect to weight. These updates for w and b are determined using the backpropagation (BP) (discussed in Section 2.4.4) method to compute the partial derivatives $\partial E/\partial w^l_{jk}$ and $\partial E/\partial b^l_j$. But to compute those partial derivatives, we introduce an intermediate quantity, δ^l_j which is an error in the j^{th} neuron of the l^{th} layer.

2.4.3 Stochastic Gradient Descent

We saw in the previous section that training a network corresponds to choosing parameters, that is, the weight and biases, that minimize the loss function. The process of choosing best the parameters for the model is often referred to as *cross-validation parameter selection* [24]. The weights and biases take the form of matrices and vectors, but at this stage we imagine them stored as a single column vector that is called p. Generally, we will suppose $p \in \mathbb{R}^g$ and write the loss function in Equation (2.4.3) as $E(p)$ to emphasize its dependence on the parameters. So $E : \mathbb{R}^g \to \mathbb{R}$. We now briefly introduce a classical method in optimization that is often referred to as *gradient descent*. The method proceeds iteratively, by computing a sequence of vectors in $\mathbb{R}^g$, with the aim of converging to a vector that will minimize the loss function. Consider our current vector as p. How should we choose an update, $\triangle p$, so that the next vector, $p + \triangle p$, represent an improvement? Using Taylor's Theorem and neglecting all but first-order terms, since we will assume that $\triangle p$ is very small, we get

$$E(p + \triangle p) \approx E(p) + \sum_{r=1}^{g} \frac{\partial E(p)}{\partial p_r} \triangle p_r. \tag{2.4.4}$$

By choosing small update $\triangle p$, we then ignore the terms with of order $|| \triangle p ||^2$. We set $\nabla E(p) \in \mathbb{R}^g$ to denote the column vector of partial derivatives, known as the *gradient*, so that

$$\Big(\nabla E(p)\Big)_r = \frac{\partial E(p)}{\partial p_r}. \tag{2.4.5}$$

Then Equation (2.4.4) becomes

$$E(p + \triangle p) \approx E(p) + \nabla E(p)^T \triangle p \tag{2.4.6}$$

To ensure that the RHS of (2.4.6) is less than the LHS we should choose $\triangle p$ so that $\nabla E(p)^T \triangle p < 0$. To be sure that this is the case, we choose $\triangle p = -\eta \nabla E(p)$ for small $\eta > 0$. This results to the update given by:

$$p \to p - \eta \nabla E(p) \tag{2.4.7}$$

where η is the learning rate. We choose an initial vector and iterate with Equation (2.4.27) until some stopping criterion has been met. Setting the learning rate is a difficult task, due to the following facts: if the learning rate is too small, an algorithm might take a long time to converge. On the other hand; larger values of η could have opposite effect, causing an algorithm to diverge. Our loss function described by Equation (2.4.3) involves a sum of individual terms that run over the training data.

The performance of ANN largely depends on the architecture of the neural network. Crucial issues in neural network modeling are the selection of inputs variables, data processing technique, network architecture design and performance measuring statistics should be carefully verified. In addition, Furthermore, ANN is a good choice as an alternative to linear forecasting models [52].

2.4.4 Backpropagation Algorithm

Neural Networks can learn their weights and biases by using a gradient descent algorithm. We are now in a position to apply the stochastic gradient descent method in order to train our network. However, computing the gradient of the cost functions requires a smart method known as BP. This method was introduced in the 1970s as a general optimization method for performing automatic differentiation of complex nested functions. However, it wasn't until 1986 [74], with the publishing of a paper by Rumelhart, Hinton, and Williams, titled "Learning Representations by Back-Propagating Errors," that the importance of the algorithm was appreciated by the machine learning community at large. Our task is to compute partial derivatives of the loss function with respect to w^l_{jk} and b^l_j by using the BP method. We have described the idea behind the stochastic gradient descent method as to exploit the structure of the cost function: because Equation (2.4.2) represents a linear combination of individual terms that runs over the training set. We therefore focus our attention on computing those individual partial derivatives.

Now, by considering a fixed training point we regard $E_{x^{(i)}}$ in Equation (2.4.2) as a function of weights and biases. So we may drop the dependence $x^{(i)}$ and simply write

$$E = \frac{1}{2} \mid\mid y - o^L \mid\mid^2 \tag{2.4.8}$$

We recall from Equation (2.4.1) that o^L is the output from the artificial neural network. Note that the dependence of the loss function E on the weights and biases arise only through o^L. In deriving the expressions for computing the partial derivatives, we introduce two further sets of variables. Firstly, we let

$$z^l = W^l o^{l-1} + b^l \in \mathbb{R}^{n_l}, \quad \text{for} \quad l = 2, 3, ..., L. \tag{2.4.9}$$

We refer to z^l_j as the *weighted input* for neuron j at layer l. Thus result to a fundamental relation of Equation (2.4.1) that propagates information through the network be written as:

$$o^l = \sigma\left(z^l\right), \quad \text{for} \quad l = 2, 3, ..., L. \tag{2.4.10}$$

Secondly, we let $\delta^l \in \mathbb{R}^{n_l}$ be defined by

$$\delta^l_j = \frac{\partial E}{\partial z^l_j}, \quad \text{for} \quad 1 \leq j \leq n_l \quad \text{and} \quad 2 \leq l \leq L \tag{2.4.11}$$

This expression, which is often called the vector of *errors* in the jth neuron at layer l. BP will give us a way of computing δ^l for every layer, and then relating those errors to the quantities of real interest $\partial E / \partial w^l_{jk}$ and $\partial E / \partial b^l_j$.

At this stage, we also need to define the Hadamard, or componentwise, a product of two vectors. If $x, y \in \mathbb{R}^n$, then $x \circ y \in \mathbb{R}^n$ is defined by $(x \circ y)_i = x_i y_i$. The Hardamard product is formed by pairwise multiplication of the corresponding components. With this notation, the following results are obtained using the chain rule.

Lemma 1

We have:

$$\delta^L = \sigma'(z^L) \circ (o^L - y) \tag{2.4.12a}$$

$$\delta^L = \sigma'(z^l) \circ (W^{l+1})^T \delta^{l+1}, \quad \text{for} \quad 2 \leq l \leq L-1 \tag{2.4.12b}$$

$$\frac{\partial E}{\partial b_j^l} = \delta_j^l, \quad \text{for} \quad 2 \leq l \leq L \tag{2.4.12c}$$

$$\frac{\partial E}{\partial w_{jk}^l} = \delta_j^l o_k^{l-1}, \quad \text{for} \quad 2 \leq l \leq L \tag{2.4.12d}$$

Proof

We begin by proving Equation (2.4.12a). The relation in Equation (2.4.10) with $l = L$ shows that z_j^L and o_j^L are connected by $o^L = \sigma(z^L)$, and hence

$$\frac{\partial o_j^L}{\partial z_j^L} = \sigma'(z_j^L). \tag{2.4.13}$$

Also, from Equation (2.4.8),

$$\frac{\partial E}{\partial o_j^L} = \frac{\partial}{\partial o_j^L}\left(\frac{1}{2}\sum_{k=1}^{n_L}\left(y_k - o_k^L\right)^2\right) = -\left(y_j - o_j^L\right). \tag{2.4.14}$$

So, applying the chain rule,

$$\delta_j^L = \frac{\partial E}{\partial z_j^L} = \frac{\partial E}{\partial o_j^L}\frac{\partial o_j^L}{\partial z_j^L} = \left(o_j^L - y_j\right)\sigma'(z_j^L), \tag{2.4.15}$$

which is the componentwise form of Equation (2.4.12a). To prove Equation (2.4.12b), we proceed as follow. Now the question arises on how the partial derivatives of layers other than the output layer can be calculated. Luckily, the chain rule for multivariate functions come to rescue again. We use the chain rule to convert z_j^l to $\left\{z_k^{l+1}\right\}_{k=1}^{n_{l+1}}$. Observe the following equation for the error term δ_j^l, using the definition represented by Equation (2.4.11),

$$\delta_j^L = \frac{\partial E}{\partial o_j^L} = \sum_{k=1}^{n_{l+1}} \frac{\partial E}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_{k=1}^{n_{l+1}} \delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial z_j^l}. \tag{2.4.16}$$

Now, from Equation (2.4.8) we know that z_k^{l+1} and z_j^l are connected via

$$z_k^{l+1} = \sum_{g=1}^{n_l} w_{kg}^{l+1}\sigma(z_g^l) + b_k^{l+1}. \tag{2.4.17}$$

Hence,

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1}\sigma'(z_j^l). \tag{2.4.18}$$

In Equation (2.4.16) this gives

$$\delta_j^l = \sigma(z_j^l)\left((W^{l+1})^T\delta^{l+1}\right)_j. \tag{2.4.19}$$

This is the componentwise form of Equation (2.4.12b). To prove Equation (2.4.12c), we note from Equation (4.4.15) and (2.4.10) that z_j^l is connected to b_j^l by the following equation

$$z_j^l = \left(W^l\sigma\left(z^{l-1}\right)\right)_j + b_j^l. \tag{2.4.20}$$

We note that z^{l-1} does not depend on b_j^l, we find that

$$\frac{\partial z_j^l}{\partial b_j^l} = 1. \tag{2.4.21}$$

Then from the chain rule,

$$\frac{\partial E}{\partial b_j^l} = \frac{\partial E}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial E}{\partial z_j^l} = \delta_j^l, \tag{2.4.22}$$

applying Equation (2.4.11). This result in Equation (2.4.12c). Finally, to obtain Equation (2.4.12d) we begin with the componentwise version of Equation (4.4.15),

$$z_j^l = \sum_{k=1}^{n_{l-1}} w_{jk}^l o_k^{l-1} + b_j^l, \tag{2.4.23}$$

which gives

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = o_k^{l-1}, \quad \text{independently of} \quad j, \tag{2.4.24}$$

and

$$\frac{\partial z_g^l}{\partial w_{jk}^l} = 0, \quad \text{for} \quad g \neq j \tag{2.4.25}$$

Thus mean Equation (3.6.4) and (2.4.25) follow the jth neuron at layer l using the weights from only the jth row of W^l, and applies these weights linearly. Then using chain rule, Equation (3.6.4) and (2.4.25) give

$$\frac{\partial E}{\partial w_{jk}^l} = \sum_{g=1}^{n_l}\frac{\partial E}{\partial z_g^l}\frac{\partial z_g^l}{\partial w_{jk}^l} = \frac{\partial E}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \frac{\partial E}{\partial z_j^l}o_k^{l-1} = \delta_j^l o_k^{l-1}, \tag{2.4.26}$$

were δ_j^l is defined by Equation (4.4.15). This completes the proof.
Applying these derivations to our architecture described in Section 2.4.4, then the weight and bias updates is

$$w_{ij}^{l_{new}} = w_{ij}^{l_{old}} - \eta\frac{\partial E}{\partial w_{jk}^L} = w_{ij}^{l_{old}} - \eta\sum_j o_j^{l-1}\delta_j^l \quad \text{and} \quad b_j^{l_{new}} = b_j^{l_{old}} - \eta\frac{\partial E}{\partial b_j^L} = b_j^{l_{old}} - \eta\sum_j \delta_j^l \tag{2.4.27}$$

where η is the learning rate. We choose an initial vector and iterate with Equation (2.4.27) until some stopping criteria has been met.

The General Algorithm

The backpropagation algorithm proceeds in the following steps, assuming a suitable learning rate η and random initialization of the parameters w_{jk}^l:

(i) **Calculate the forward phase** for each input-output pair (x^d, y^d) and store the results $\hat{y}^d$ and o_j^l for each node j in the layer l by proceeding from layer 1, the input layer, to layer L, the output layer.

(ii) **Calculate the backward phase** for each input-output pair (x^d, y^d) and store the results $\frac{\partial E}{\partial w_{jk}^l}$ for each weight w_{jk}^l connecting to node k in the layer $l-1$ to node j in the layer l proceeding from layer L , the output layer, to layer 1, the input layer.

- Evaluate the error term for the final layer δ_1^L by using Equation (2.4.16)
- Backpropagate the error terms for the hidden layers δ_j^k, working backward from the final hidden layer $l = L-1$, by repeatedly using Equation (2.4.19)
- Evaluate the partial derivatives of the individual error E_d with respect to w_{jk}^l using Equation (2.4.26)

(iii) **Combine the individual gradients** $\frac{\partial E}{\partial w_{jk}^l}$ for each input-output pair to get the total gradient $\frac{\partial E(X,w)}{\partial w_{jk}^l}$ for the entire set of input-output pairs $X = \left\{(x^{(1)}, y^{(1)}), ..., (x^{(N)}, y^{(N)})\right\}$

(iv) **Update the weights** according to the learning rate η and total gradient $\frac{\partial E(X,w)}{\partial w_{jk}^l}$ by using Equation (6.2)

2.5 History

A full review of the historical developments in neural network research is beyond the scope of this thesis. This section provides a brief overview of some events in the network history by focusing on some important breakthroughs up to 1994. For a thorough survey, [29] is recommended.

In 1943 McCulloch and Pitts began to use mathematics to describe the mechanisms of the human brain [29]. This allowed them to model logical operations (e.g. AND, OR or NOT) using their neural network. Their work can be seen as the beginning of the modern era in the artificial neural network field [29], [21]. McCulloch and Pitts have developed an artificial neuron model (known as the McCulloch and Pitts model) with a simple threshold activation function where the output is either zero or one [29]. This model of a neuron is still widely used [21].

Hebb suggested in 1949 that the synaptic connections inside the brain change constantly as a person gains experience [67]. In other words, synapses are either strengthened or weakened depending on whether neurons on either side of the synapse are continuously activated simultaneously or not. This led to the development of the first procedure for learning artificial neural networks where the synapse is modified by changes in synaptic weights [67].

In the late 1950s, Rosenblatt examined how the brain distinguishes different types of stimuli, which led to the conceptual development of the *perceptron*. The perceptron, using a singleMcCulloch and Pitts neuron is a classifier of patterns that can assign input patterns to one of

two classes. The perceptron can solve classification problems with different class numbers depending on the number of neurons included [29].

In 1969 Minsky and Papert published a book called "Perceptrons" in which the perceptron was criticized [21]. In the book, they pointed [48] two fundamental flaws, the calculation of topological functions of connectedness and the calculation of parity, which Rosenblatt's perceptrons [13] could not solve. This led to the inability of single-layer perceptors to solve classification problems that can not be separated linearly [29]. This was demonstrated by Misky and Papert in 1969, who also raised the issue of the multi-layer perceptron issue of *credit assignment*. The results of the analysis by Misky and Papert [47] led them to conclude that although perceptrons were "interesting" to study, perceptrons and their possible extensions were "sterile" research directions. The publication of their book has led to a reduction in research into artificial neural networks [21, 29].

Influenced by brain studies (i.e. brain maps), Kohonen developed the *Self-Organizing Map (SOM)* artificial neural network type. This SOM uses an unsupervised learning algorithm for applications in data mining, image processing, and visualization. As a basic description, the structured lattice architecture of the SOM maps high-dimensional input to low-dimensional representation. The network of Kohonen is one of the most popular unsupervised artificial neural networks. In the same year, John Hopfield built a bridge between physics and neural computing. Hopfield connected the artificial neural networks to the field of physics in 1982 [29]. The Hopfield networks are connected in such a way that they start in a random state and then move to the final stable state [67].

The Boltzmann machine was invented in 1985. As the name suggests, Ludwig Boltzmann's work in thermodynamics has been an inspiring source. This neural network utilizes a stochastic learning algorithm based on properties of the Boltzmann distribution.

The discovery of the back-propagation algorithm in 1986 (see Section 2.4.4) was crucial for the regeneration of neural networks. In 1974 Werbos solved the problem of credit assignment for multi-layered perceptrons with the invention of the back-propagation algorithm [67]. Rummelhart, Hinton and Williams received the credit, but it showed that Werbos already introduced the error back-propagation in his Ph.D. thesis in 1974. Research on neural networks became important again with the invention of the back-propagation algorithm and has since been one of the larger areas of machine learning [60].

The radial base (RBF) network was developed in 1988 by Broomhead and Lowe [29]. Radial function networks are universal function approximation and can be used in areas such as classification of patterns, function approximation or regularization [67].

Chapter 3

Recurrent Neural Networks

3.1 Introduction

This chapter describes the Recurrent Neural Network (RNN) model in its full scope. Section 3.2 and 3.2.1 provides brief introductory on unfolding computational graph for RNN and the concept of overshooting in RNN, while Section 3.3 describes *backpropagation-through-time* process and how it works in an RNN architecture. Finally, Section 3.4 detail the concept of *exploding and vanishing gradient*, while Section 3.4 discusses the solution on how to resolve the issue of vanishing and exploding gradients.

RNNs are a class of supervised machine learning models made up of artificial neurons with one or more feedback loops [29]. In order to get a deeper understanding of the functioning and composition of RNN, we introduce a conceptual trick which is called the correspondence principle between equations, architectures and local algorithms. The Neural Network (NN) equations can be graphically represented using an architecture that represents the individual layers of the network in the form of nodes. In addition, the edges are represented by matrices between the layers. This correspondence is most effective in combination with the local optimizing algorithms that provide the basis for the training of the NNs. For example during training, forward and backward flow provides locally available information which is used during back-propagation to calculate the derivatives of the NN error functions [63].

3.2 Unfolding Computation Graph for RNN

A computational graph is a directed graph used to formalize the structure of a set of computations, such as mapping inputs and parameters to outputs and loss. In this section we explain the concept of unfolding a recursive or recurrent computation into a cyclic graph typically corresponding to a loop of events. Another interesting property of unfolding RNN structure is the sharing of parameters across a deep network structure.

For example, let us consider a dynamical system where the present state is a function of the previous state. It can be expressed compactly as follows:

$$s^{(t)} = f\left(s^{(t-1)}; \theta\right) \tag{3.2.1}$$

where $s^{(t)}$ is the state of the system at time t. This is a recursive or recurrent definition: the state at time $'t'$, $s^{(t)}$ is a function (f) of the previous state $s^{(t-1)}$, parameterized by θ. This equation can be *unfolded* as follows:

$$s^{(3)} = f\Big(s^{(2)};\theta\Big) = f\Big(f\Big(s^{(1)};\theta\Big);\theta\Big). \tag{3.2.2}$$

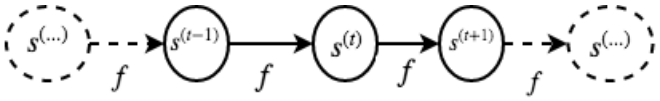


Figure 3.1: The classical dynamical system is described by Equation (3.2.3), illustrated as an unfolded computational graph. Each node represents the state at some time t and the function f maps the state at t to state at $t+1$. The same parameters (the same value of θ) are used for all time steps.

We now consider a slightly more complex system, whose state not only depends on the previous state, but also on an external signal $x^{(t)}$

$$s^{(t)} = f\Big(s^{(t-1)};x^{(t)};\theta\Big) \tag{3.2.3}$$

where we observe the state now containing information about the whole past sequence.

Now we introduce the basic recurrent neural network (RNN) in state form. We consider a simple RNN that has three layers which are input, recurrent hidden, and output layers, as represented in Figure 3.2. The input layer has N input units. The input to this layer is a sequence of vectors with time index t such as $\{...,x^{(t-1)},x^{(t)},x^{(t+1)},...\}$, where $x \in \mathbb{R}^d$. The input units in a fully connected RNN connect to hidden units in the hidden layer, where the connections are defined by weight matrix W^{xs}. The hidden layer has M hidden units $s^{(t)} \in \mathbb{R}^M$, that are connected to each other through time with recurrent connections, (see Figure 3.2). The recurrent network can be described as a *dynamical system* by the non-linear matrix equations described below:

$$s^{(t)} = f_s\Big(W^{xs}x^{(t-1)} + W^{ss}s^{(t-1)}\Big) \tag{3.2.4}$$

where $s^{(t)}$ represents the hidden layer which defines the state space or "memory" of the system and $f_y(.)$ is the hidden layer *tanh* activation function defined in Section 2.3.1 by Equation (2.3.3). Here W^{xs} is the weight matrix between the input and the hidden layer and W^{ss} is the matrix of recurrent weights between the hidden layer and itself as adjacent time steps. The hidden units are connected to the output layer with weighted connections W^{sy}. The output layer has l units $y^{(t)} = \Big(y^{(1)},y^{(2)},...,y^{(l)}\Big)$ that are computed by Equation (3.2.5)

$$y^{(t)} = f_y\Big(W^{sy}s^{(t)}\Big) \tag{3.2.5}$$

where $f_s(.)$ is the output layer *sigmoid* activation function defined in Section 2.3.1 by Equation 2.3.4. Below are details associated with each parameter in the network described by Equations (3.2.4) and (3.2.5).

- $x^{(1)},...,x^{(t-1)},x^{(t)},x^{(t+1)},...,x^{(T)}$: the input price vector with total length T.
- $s^{(t)} = f_s\Big(W^{xs}x^{(t-1)} + W^{ss}s^{(t-1)}\Big)$: the relationship to compute the hidden layer output features at each time-step t.

  - $x^{(t-1)} \in \mathbb{R}^d$, input price vector at time $t-1$.
  - $W^{xs} \in \mathbb{R}^{D_s \times d}$: weight matrix used to condition the input price vector, $x^{(t-1)}$.
  - $W^{ss} \in \mathbb{R}^{D_s \times D_s}$: weight matrix used the output to previous time-step, $s^{(t-1)}$.
  - $s^{(t-1)} \in \mathbb{R}^{D_s}$; output of the non-linear function at the previous time-step, $t-1$. $s^{(0)} \in \mathbb{R}^{D_s}$ is an initialization vector for the hidden layer at time step $t = 0$.
  - f_s: non-linear function (*tanh* here).

- $y^{(t)} = f_y\left(W^{sy}s^{(t)}\right)$: the output over the price at each time-step t. Essentially, $y^{(t)}$ is the next predicted price given the output of the hidden layer $s^{(t-1)}$ and the last observed price vector $x^{(t-1)}$. Here $W^{sy} \in \mathbb{R}^{|V| \times D_h}$ and $y^{(t)} \in \mathbb{R}^{|V|}$ where $\mid V \mid$ is the size of the output vector.

The dynamics of the network represented by the leftmost as shown in Figure 3.2 across time steps can be visualized by *unfolding* it as in Figure 3.2. Given the graphical computation shown in Figure 3.2, the network can be interpreted not as the recursive structure, but rather as a deep network with one layer per time step and shared weights across time steps.

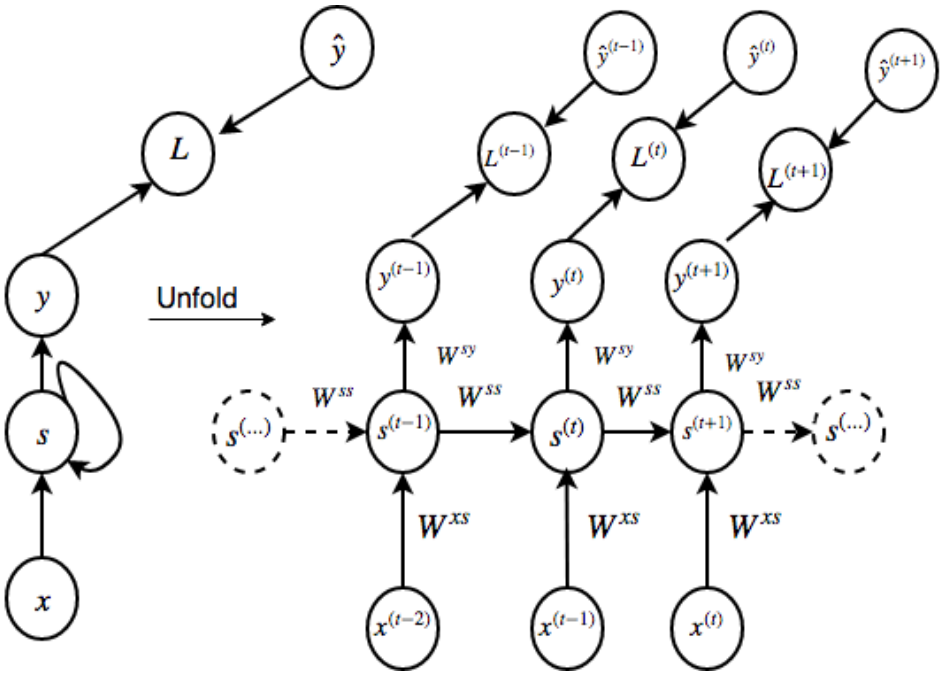


Figure 3.2: The computational graph to compute the training loss of the recurrent network that maps an input into a sequence of x values to a corresponding sequence of output y values. A loss function L measures how far each y is from the corresponding training target $\hat{y}$ (left). The RNN and its loss draw with recurrent connections (right).

The approximation step of the finite unfolding truncates the unfolding after some time steps, for example, we choose any hidden state backward $(t-2, t-3, t-4, t-n)$. The important question to solve is the determination of the correct amount of past information to include in the model of $y^{(t+1)}$. Typically, you start with a given truncation length, then you can observe the individual error of the outputs (e.g. $y^{(t-1)}, y^{(t)}, y^{(t+1)}$ see Figure 3.2) computed by Equation (3.2.5) which usually decrease from left ($y^{(t-1)}$) to right ($y^{(t)}$). The reason for this

observation is that the leftmost output $y^{(t-1)}$ is computed only by the most past external information $x^{(t-2)}$ but it uses the additional information of a previously hidden state, in this case that would be internal state $s^{(t-2)}$. By superposition of more and more information, the loss function will decrease until a minimal loss is achieved.

Furthermore, overshooting has an application for the learning itself. Summarizing, it should be noted, that overshooting generates additional valuable forecast information about the underlying analyzed dynamical system. In this thesis, we will use a type of RNN architecture to perform a supervised learning for short term and long term tasks. So, how does one train recurrent neural networks? Section 3.3 and 3.4 briefly explore widely used learning techniques often referred to as *backpropagation-through-time* and we also discuss the issue of the *vanishing gradients* in case of a standard RNN.

3.2.1 Overshooting in Recurrent Neural Network

In terms of application, we often observe that recurrent neural networks tend to focus on only the most central internal inputs in order to explain the dynamics. A generalization of the network in Figure 4.3 is the extension of the autonomous recurrence in the future directions (here t+2, t+3, t+4) this is called *overshooting* (see Figure 4.3). In order to describe the development of the dynamics in one of these future time steps adequately, matrix W^{ss} must be able to transfer information over time [25]. If this leads to good performance in terms

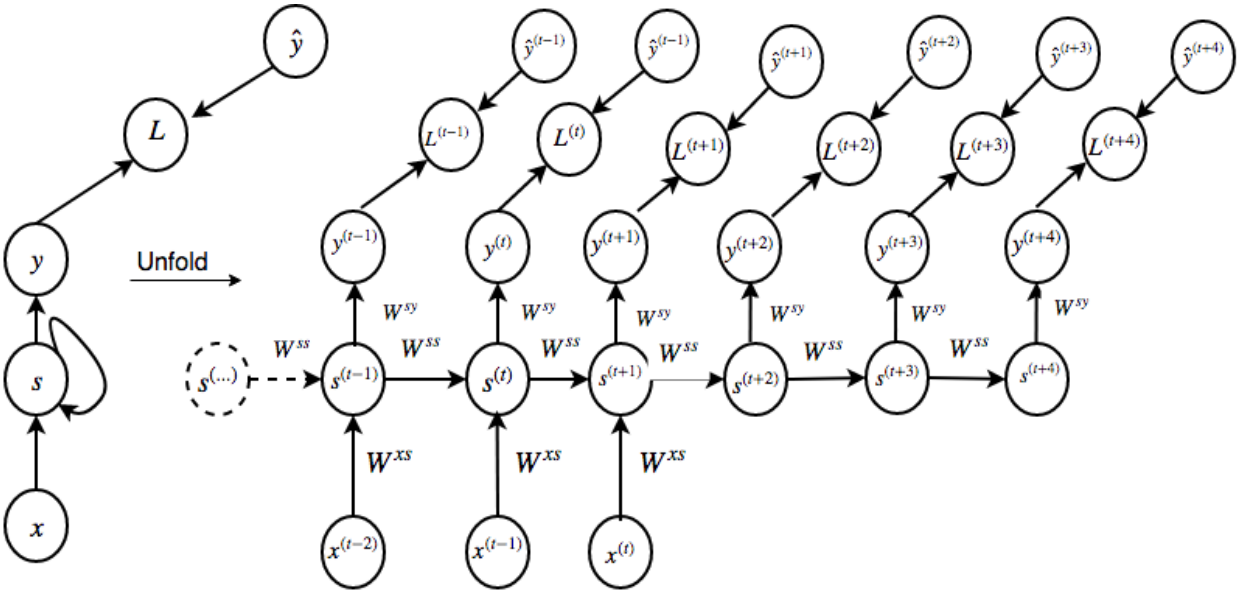


Figure 3.3: RNN incorporating overshooting

of forecasting, then we get as an output a whole sequence of forecasts. In most cases, it is applicable in decision support systems (e.g trading systems in finance).

In the following, we briefly show how overshooting can be realized and we will analyze its properties. First, we discuss how far should the overshooting be extended in the future to attain good predictions. We iterate the following: train the model until the error loss converges to a global minimum. If overfitting occurs, include the next output (here $y^{(t+k+1)}$) and train it again [25]. Typically through the process of *cross-parameter validation selection*, we observe the following interesting phenomenon: If new prediction $y^{(t+k+1)}$ can be learned by

the network, the error for this newly activated time horizon (daily and weekly) will decrease. In this thesis, we focus on predicting one step ahead for the overshooting architecture.

3.3 Backpropagation Through Time

The back-propagation-through-time (BPTT) learning algorithm is a natural extension of standard back-propagation performing gradient descent on an unfolded with time network. To gain some intuition for how BPTT algorithm behaves, we provide an example of how to compute gradients by BPTT for RNN equations shown by Equations (3.2.4) and (3.2.5). The BPTT is an algorithm that computes the chain rule, with a specific order of operations that is highly efficient.

A brief introduction on some important operations used in this chapter are introduced here: Let x be a real number, and let f and g both be functions mapping from real number to a real number. Suppose that $y = g(x)$ and $z = f(g(x)) = f(y)$. Then the chain rule states that

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{3.3.1}$$

In case of generalizing beyond the scaler case. Suppose that $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, g maps from $\mathbb{R}^m$ to $\mathbb{R}^n$ and f maps from $\mathbb{R}^n$ to $\mathbb{R}$. If $\mathbf{y} = g(\mathbf{x})$ and $z = f(\mathbf{y})$, then

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{3.3.2}$$

In vector notation, this may be equivalent written as

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \nabla_{\mathbf{y}} z, \tag{3.3.3}$$

where $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is the $n \times m$ Jacobian matrix of g. From this, we note that the gradient of a variable x can be obtained by multiplying a Jacobian matrix $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ by a gradient $\nabla_{\mathbf{y}} z$. The BPTT algorithm consists of performing such Jacobian-gradient product for each operation as shown below. The total loss for a given sequence of x values paired with a sequence of y values would then be just the sum of the losses over all the time steps. Given a sequence of T input vectors. The loss function evaluates the performance of the network by comparing the output $y^{(t)}$ with the corresponding target $\hat{y}^{(t)}$ defined as

$$L = \frac{1}{T}\sum_t^T L^{(t)} \tag{3.3.4}$$

where $L^{(t)}$ is defined as

$$L^{(t)} = \frac{1}{2}\left(\hat{y}^{(t)} - y^{(t)}\right)^2 \tag{3.3.5}$$

where $\hat{y}^{(t)}$, $y^{(t)}$ are actual and predicted values respectively and T is the final time step. Then the gradient $\nabla_{y^{(t)}} L$ on the outputs at time step t as:

$$\nabla_{y^{(t)}} L = \frac{\partial L^{(t)}}{\partial y_i^{(t)}} = y^{(t)} - \hat{y}^{(t)} \tag{3.3.6}$$

the above Equation (4.4.3) is obtained given the activation function applied to $y^{(t)}$ as *sigmoid* function defined in Section 2.3.1 by Equation (2.3.4). Now, starting from the end of the sequence propagating backward. At the final time step T, $s^{(T)}$ only has $y^{(T)}$ as a descendent, resulting in the following gradient

$$\nabla_{s^{(\tau)}} L = W^{sy\top} \nabla_{y^{(\tau)}} L \tag{3.3.7}$$

We now back-propagate gradients through time by iterating backward in time, from $t = T-1$ down to $t = 1$, noting that $s^{(t)}$ $\left(\text{for } t < T\right)$ has descendent both $y^{(t)}$ and $s^{(t+1)}$. Its gradient is given by:

$$\begin{aligned} \nabla_{s^{(t)}} L &= \left(\frac{\partial s^{(t+1)}}{\partial s^{(t)}}\right)^{\top} \left(\nabla_{s^{(t+1)}} L\right) + \left(\frac{\partial y^{(t)}}{\partial s^{(t)}}\right)^{\top} \left(\nabla_{y^{(t)}} L\right) \\ &= W^{ss\top} \operatorname{diag}\left(1 - \left(s^{(t+1)}\right)^2\right) \left(\nabla_{s^{(t+1)}} L\right) + W^{sy\top} \left(\nabla_{y^{(t)}} L\right) \end{aligned} \tag{3.3.8}$$

where $\operatorname{diag}\left(1 - \left(s^{(t+1)}\right)^2\right)$ indicates the diagonal matrix containing the elements $1 - \left(s^{(t+1)}\right)^2$. This is the Jacobian of the hyperbolic tangent associated with the hidden unit i at time $t+1$. In order to transport the error through time from time-step T back to time-step t we can have

$$\frac{\partial s^{(T)}}{\partial s^{(t)}} = \prod_{i=t+1}^{T} \frac{\partial s^{(i)}}{\partial s^{(i-1)}}. \tag{3.3.9}$$

We consider Equation (3.3.9) as a Jacobian matrix for the hidden state parameter in Equation (3.2.4) as

$$\frac{\partial s^{(T)}}{\partial s^{(t)}} = \prod_{i=t+1}^{T} \frac{\partial s^{(i)}}{\partial s^{(i-1)}} = \prod_{i=t+1}^{T} W^{ss\top} \operatorname{diag}\left(1 - \left(s^{(t+1)}\right)^2\right), \tag{3.3.10}$$

because $s \in \mathbb{R}^{D_n}$, each $\partial s^{(i)} / \partial s^{(i-1)}$ is the Jacobian matrix for s:

$$\frac{\partial s^{(i)}}{\partial s^{(i-1)}} = \left[\frac{\partial s^{(i)}}{\partial s^{(i-1,1)}} \cdots \frac{\partial s^{(i)}}{\partial s^{(i-1,D_n)}}\right] = \begin{bmatrix} \frac{\partial s^{(i,1)}}{\partial s^{(i-1,1)}} & \cdots & \frac{\partial s^{(i,1)}}{\partial s^{(i-1,D_n)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s^{(i,D_n)}}{\partial s^{(i-1,1)}} & \cdots & \frac{\partial s^{(i,D_n)}}{\partial s^{(i-1,D_n)}} \end{bmatrix}.$$

The derivation shown by Equation (2.2) which describe the backward in time iteration of the loss function L with respect to hidden state $s^{(t)}$ is adopted from Bengio book [34]. We take a note of the long term and short term contribution of the hidden state over time in the network. The long-term dependency refers to the contribution of the inputs with corresponding of inputs with corresponding hidden states at time $t << T$. The dynamics represented by Figure 3.4 shows that as the network makes progress over time, the contribution of the inputs $x^{(t-1)}$ at discrete time $t-1$ vanishes through time to the time-steps $t+1$ (dark grey in layers decay to higher light grey). However, on the other side loss function $L^{(t+1)}$ with respect to hidden state $s^{(t+1)}$ at time $t+1$ in BPTT is more than previous time-steps. The issue of exploding and vanishing gradients is briefly discussed in Section 3.4.

Now, the gradients on the internal nodes of the computational graph are obtained first, then the gradients on the parameter nodes. Since the parameters are shared across many time

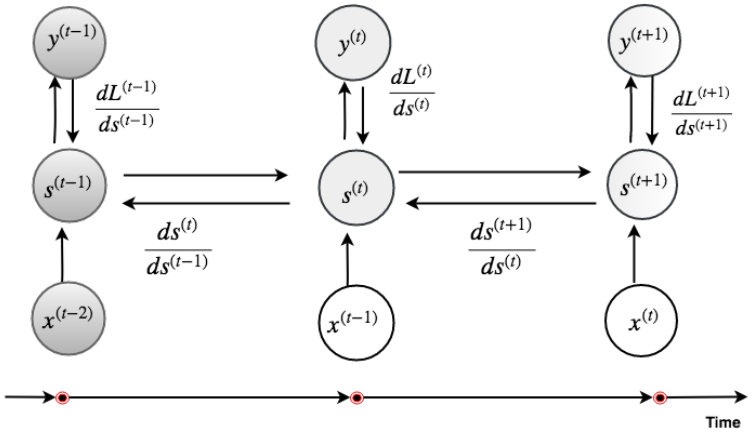


Figure 3.4: As the network receives new input over time, the sensitivity of units decay (lighter grey shades in layer) and the back-propagation through time (BPTT) overwrites the activation function in the hidden units.

steps, we carefully denote calculus operations involving these variables. We first find the derivative of the error function with respect to parameter W^{sy} which is present in the function $\hat{y}$ defined by equation 3.2.3.

Consider Equations (3.2.4) and (3.2.5) at time step t. To compute the RNN errors, dL/dW^{xs}, dL/dW^{ss}, dL/dW^{sy}, we sum error at each time step. That is, dL_t/dW^{xs}, dL_t/dW^{ss}, dL_t/dW^{sy} for every time step,t, is computed and accumulated

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L^{(t)}}{\partial W} \tag{3.3.11}$$

where W are network parameters $\left\{W^{xs}, W^{ss}, W^{sy}\right\}$. The error for each time-step is computed through applying the chain rule differentiation to Equations (3.2.4) and (3.2.5). Notice $ds^{(t)}/ds^{(k)}$ refers to the partial derivative of $s^{(t)}$ with respect to all previous k time-steps.

$$\frac{\partial L^{(t)}}{\partial W} = \sum_{k=1}^{t} \frac{\partial L^{(t)}}{\partial y^{(t)}} \frac{\partial y^{(t)}}{\partial s^{(t)}} \frac{\partial s^{(t)}}{\partial s^{(k)}} \frac{\partial s^{(k)}}{\partial W}. \tag{3.3.12}$$

Using the notation described at the beginning of this section, the partial derivative of the loss function with respect to each parameter is given by the following expressions;

$$\begin{aligned}\nabla_{W^{sy}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial y_i^{(t)}} \right) \nabla_{W^{sy}} y_i^{(t)} \\ &= \sum_t \left(\nabla_{y^{(t)}} L \right) {s^{(t)}}^\top \\ &= \sum_t \left(y^{(t)} - \hat{y}^{(t)} \right) {s^{(t)}}^\top \end{aligned} \tag{3.3.13}$$

$$\begin{aligned}\nabla_{W^{ss}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial s_i^{(t)}} \right) \left(\nabla_{W^{ss}} s_i^{(t)} \right) \\ &= \sum_t \operatorname{diag} \left(1 - \left(s^{(t)} \right)^2 \right) \left(\nabla_{s^{(t)}} L \right) {s^{(t-1)}}^\top \end{aligned} \tag{3.3.14}$$

$$\begin{aligned}\nabla_{W^{xs}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial s_i^{(t)}} \right) \left(\nabla_{W^{xs}} s_i^{(t)} \right) \\ &= \sum_t \operatorname{diag} \left(1 - \left(s^{(t)} \right)^2 \right) \left(\nabla_{s^{(t)}} L \right) {x^{(t)}}^\top \end{aligned} \tag{3.3.15}$$

where $\nabla_{s^{(t)}} L$ is defined by Equation (3.3.8). Note that we do not need to compute the gradient with respect to $x^{(t)}$ for training because it does not have any parameters as ancestors in the computational graph defining loss.

3.4 Vanishing and Exploding Gradient Problem

As introduced in Bengio [51], the problem with *exploding gradient* refers to the exponential decrease in gradient magnitudes during training. Such events are caused by the explosion of the long-term dependencies. The *vanishing gradient* refers to the exponential reduction of the gradients as they are propagated over time, making it impossible for the model to learn a correlation between events that are temporarily far. temporally distant events. There are two reasons for that

(i) Standard non-linear functions such as the sigmoid function have a gradient which is almost everywhere close to zero.

(ii) The magnitude of the gradient is continuously multiplied by the recurrent matrix as it is backpropagated through time.

Putting Equations (3.3.11), (3.3.12) and (3.3.10) together, we have the following relation

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial L^{(t)}}{\partial y^{(t)}} \frac{\partial y^{(t)}}{\partial s^{(t)}} \left(\prod_{i=t+1}^{T} \frac{\partial s^{(i)}}{\partial s^{(i-1)}} \right) \frac{\partial s^{(k)}}{\partial W} \tag{3.4.1}$$

The norm of the partial gradient at each time-step, t, is therefore, calculated through the relationship shown in Equation (3.4.2) shows a bound for the norm

$$\left\| \frac{\partial \mathbf{s}^{(t+1)}}{\partial \mathbf{s}^{(t)}} \right\| \leq \left\| W^{ss\top} \right\| . \left\| \text{diag}\left(1 - \left(\mathbf{s}^{(t+1)}\right)^2\right) \right\| \leq \beta_W \beta_s. \tag{3.4.2}$$

Equation (3.4.3) shows the norm of the Jacobian matrix relationship in Equation (3.3.10). Here, β_W and β_s represent the upper bound values for the two matrix norms. The norm of both matrices equals to their L_2-norm. The norm of $f_s(s^{(i-1)})$ can only be as large as 1 given the sigmoid non-linearity function.

$$\left\| \frac{\partial \mathbf{s}^{(T)}}{\partial \mathbf{s}^{(t)}} \right\| = \left\| \prod_{i=t+1}^{T} \frac{\partial s^{(i)}}{\partial s^{(i-1)}} \right\| \leq (\beta_W \beta_s)^{T-t}, \tag{3.4.3}$$

The exponential term $(\beta_W \beta_s)^{T-t}$ can easily become very small or lager when $(\beta_W \beta_s)$ is much smaller or lager than 1 and $T - t$ is sufficiently large. Recall a larger $T - t$ evaluates the loss function due to faraway prices. The contribution of faraway prices to predicting the next price at time-step t diminishes when the gradient vanishes early on. During experimentation, once the gradient value grows extremely large, it causes an overflow (i.e. NaN) or we might see irregular oscillations in training cost when we plot the learning curve which is easily detectable at runtime. This issue is called the *Gradient Exploding Problem*.

Solution to the Exploding and Vanishing Gradients

A brief summary about the nature of the vanishing gradients problem and how it manifests itself in deep neural networks is described below [11]. To solve this problem we focus on a simple and practical heuristic. A solution to fix this is to apply gradient clipping which was proposed by Thomas Mikolov [51]; which places a predefined threshold on the gradients to prevent it from getting too large, and by doing this it doesn't change the direction of the gradients it only changes its length. That is, whenever gradients reach a certain threshold, they are set back to a small number as shown in the algorithm below:

Algorithm 1 Psudo-code for norm clipping in the gradients whenever they explode.

Require: $\hat{g} \leftarrow \frac{\partial E}{\partial W}$
  if $|| \hat{g} || \geq$ **then**
    $\hat{g} \leftarrow \frac{threshold}{||\hat{g}||} \hat{g}$
  end if

Figure 3.5 shows the effects of gradient clipping. It shows the decision surface of a small recurrent neural network with respect to its W matrix and its bias terms, b. The surface error of a single hidden unit recurrent network that emphasizes the existence of high curvature walls. The solid lines show standard trajectories which may be followed by gradient descent. Using dashed arrow the diagram shows what would happen if the gradients are rescaled to a fixed size when its norm is above a threshold. To solve this problem of vanishing gradients, we introduce two techniques. The first technique is that instead of initializing W^{ss} randomly, start off from an identity initialization. The second technique is to use the Rectified Linear Unit (ReLU) discussed in Section 2.3.1 instead of the sigmoid function. The derivative of a ReLU is either 0 or 1. This results in gradients on top to flow through the neurons whose derivative is 1 without getting attenuated while propagating back through time-steps.

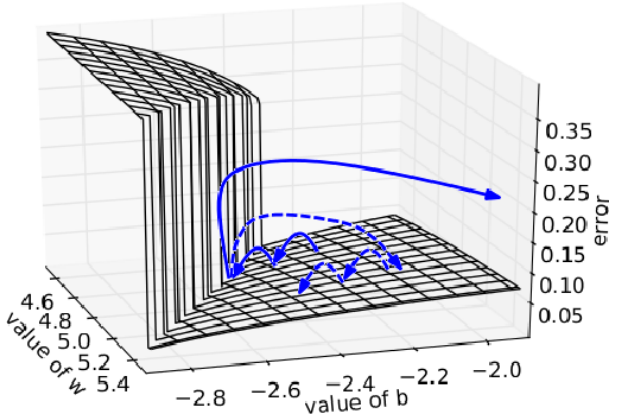


Figure 3.5: gradient explosion clipping visualization, adopted from [51].

3.5 Gradient-based Learning Methods

Gradient descent (GD) is a simple and popular optimization method in deep learning. The basic idea around this optimizing method is to adjust the weights of the model by finding the derivative of the error function with respect to each member of the weight matrices in a model [12]. Gradient descent is used to update weights parameters in the direction of the gradients of the error function until a global minimum is reached. The GD is also as batch GD, as it performs redundant computation for large datasets in each optimization iteration to perform a single update.

$$W^{(t+1)} = W^{(t)} - \frac{\eta}{T}\sum_{t=1}^{T}\frac{\partial L^{(t)}}{\partial W} \tag{3.5.1}$$

where T is the size of the training set and η is the learning rate and W are network parameters $\left\{W^{xs}, W^{ss}, W^{sy}\right\}$. This approach is not suitable for online training (i.e. training the models as input arrives) since it is computationally expensive for very large datasets.

3.5.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a generalization of GD that is widely used in the field of machine learning applications [57]. The SGD is scalable, robust, and performs quite well across many different domains with smooth, convex loss functions and non-convex ones. Despite the superfluous computation in GD, SGD simply does away with the expectation in the update and computes the gradients of the parameters using a single or few training examples. Parameters in W are updated as

$$W^{(t+1)} = W^{(t)} - \frac{\eta}{T}\sum_{t=1}^{T}\frac{\partial L^{(t)}(W, x^{(t)}, y^{(t)})}{\partial W} \tag{3.5.2}$$

with a pair $(x^{(i)}, y^{(i)})$ from the training set. Such occurrence update causes fluctuation in the loss function outputs, which helps the SDG to explore the problem landscape with higher diversity with the hope of finding parameters in W that will minimize the loss function.

3.5.2 Adaptive Gradient Algorithm (Adagrad)

The first adaptive learning rate method, proposed by Duchi [17], is Adagrad. Unlike the previously discussed approach, Adagrad maintains a different learning rate for each parameter. Some of its essential properties [53] are as follow :

- It adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rate) for parameters associated with frequently occurring features, and larger update (i.e. high learning rates) for parameters which are associated with infrequent features. For this reason, it is suitable for dealing with sparse data.

In Section 3.5, we performed an update for all parameters W at once at every parameter $W^{(i)}$ using the same learning rate η. However, Adagrad uses different learning for every parameter $W^{(i)}$ at each time step t. We first show Adagrad's per-parameter update, which is then vectorized. For brevity, let's denote $g^{(t)}$ as the gradient at time step t, $g^{(t,i)}$ is then the partial derivative of the objective function w.r.t to the parameter $W^{(i)}$ at time step t.

$$g^{(t,i)} = \nabla_W L(W^{(t,i)}), \tag{3.5.3}$$

where $L(W^{(t,i)})$ is the objective function. In its update rule, Adagrad modifies the general learning rate η at each time step t for every parameter $W^{(i)}$ based on the past gradients that have been computed for $W^{(i)}$:

$$W^{(t+1,i)} = W^{(t,i)} - \eta \left(\frac{1}{\sqrt{G^{(t,ii)} + \epsilon}} \cdot g^{(t,i)} \right) \tag{3.5.4}$$

where $G^{(t,ii)}$ is the ii element of the matrix $G^{(t)}$ [56], while ϵ is a smoothing term that avoids division by zero (usually set to an order of 10^{-8}). As $G^{(t)}$ contains the sum of the squares of the past evaluated gradients w.r.t. to all the parameters W along its diagonal, we can now vectorize our implementation by performing a matrix-vector product $\odot$ between $G^{(t)}$ and $g^{(t)}$:

$$W^{(t+1)} = W^{(t)} - \eta \left(\frac{1}{\sqrt{G^{(t)} + \kappa}} \odot g^{(t)} \right) \tag{3.5.5}$$

where κ is a diagonal matrix with ϵ value along the diagonal. One of the main benefits of Adagrad optimizing algorithm is that it eliminates the need for manually tuning the learning rate. Most implementations use a default value of 0.01 and train the model with a fixed value of the learning rate. This thesis applies two main optimization method, *Stochastic Gradient Descent*, and *Adaptive Gradient Algorithm* which have been briefly explained under Section 3.5.1 and 3.5.2 respectively. The following Section 3.6 describes the extension of RNN, which is often referred as Long Short-Term Memory (LSTM).

3.6 Long Short-Term Memory Neural Network

Recurrent dynamic connections can improve the performance of neural networks by taking advantage of their ability to understand sequential dependencies. However, the memory produced from these connections are too large which can limit the algorithms employed for training RNNs. Recall the issue mentioned about the result for the conventional RNN finding difficult to train because of the vanishing and exploding gradient during the training phase. The gradient disappears and explodes during the training phase, causing the network to fail to learn long term sequential dependencies in data [8]. The following model known as *long short-term memory* (LSTM) RNNs is designed to tackle this problem. The LSTM was

developed by Hochreiter and Schmidhuber in 1997 [32], and it is one of the most popular and efficient methods for reducing the vanishing and exploding gradient [31]. One of the advantages with the LSTM model compared to usual recurrent neural network model is the ability to capture autoregressive structures of arbitrary length. The LSTM contains special units called *memory block* in the recurrent hidden layer,This approach changes the hidden unit structure from "sigmoid" or "tanh" to memory cells in which the gates control their inputs and outputs. These gates modulate the flow of information to hidden neurons and preserve extracted features from previous timestep [31, 41].

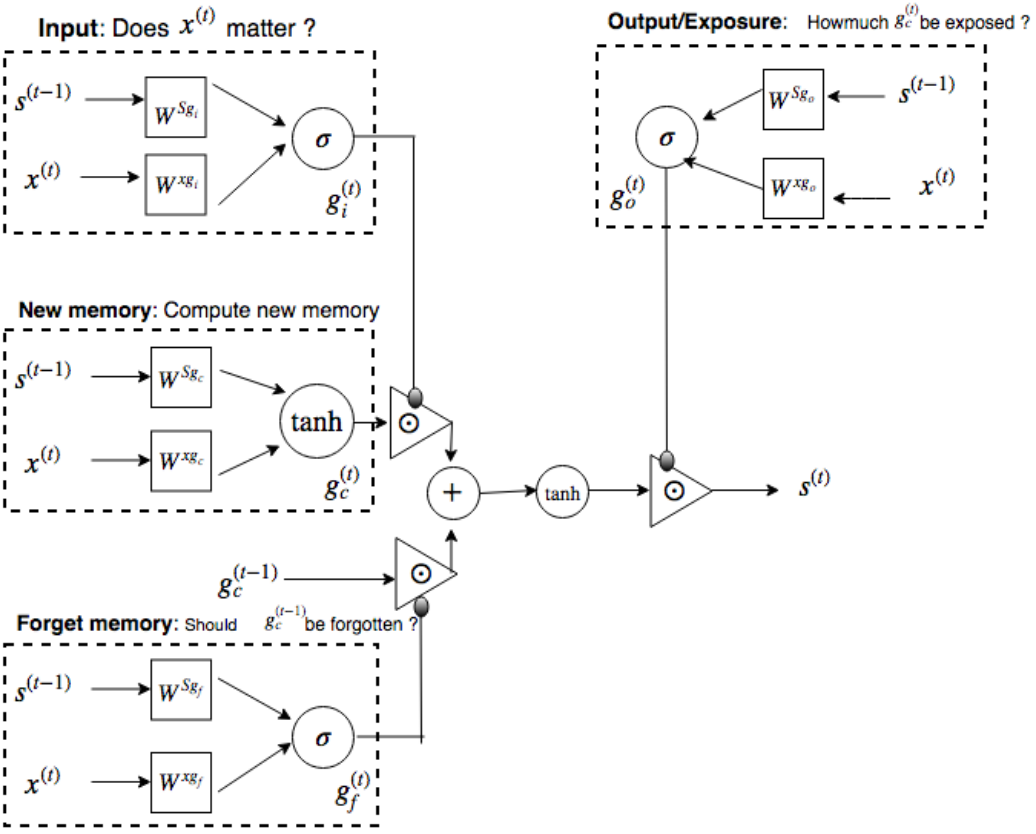


Figure 3.6: The detailed internals of an LSTM.

3.6.1 Conventional LSTM

A typical LSTM cell consists of *input*, *output*, *forget gate* and a cell activation component as shown in Figure 3.6. The input gate controls the input activation flow into the memory cell. The output gate controls cell activation output flow to the rest of the network. Later, the *forget gate* was added to the memory block [23], this type of gate is responsible in deciding upon which information will be forgotten in the cell state,

$$g_i^{(t)} = \sigma\left(W^{xg_i}x^{(t-1)} + W^{Sg_i}s^{(t-1)}\right) \quad \text{(Input gate)} \tag{3.6.1}$$

$$g_f^{(t)} = \sigma\left(W^{xg_f}x^{(t-1)} + W^{Sg_f}s^{(t-1)}\right) \quad \text{(Forget gate)} \tag{3.6.2}$$

$$g_o^{(t)} = \sigma\left(W^{xg_o}x^{(t-1)} + W^{Sg_o}s^{(t-1)}\right) \quad \text{(Output/Exposure gate)} \tag{3.6.3}$$

$$\hat{g}_c^{(t)} = \tanh\left(W^{xg_c}x^{(t-1)} + W^{Sg_c}s^{(t-1)}\right) \quad \text{(New memory cell)} \tag{3.6.4}$$

$$g_c^{(t)} = g_f^{(t)} \odot \hat{g}_c^{(t-1)} + g_i^{(t)} \odot \hat{g}_c^{(t)} \quad \text{(Final memory cell)} \tag{3.6.5}$$

$$s^{(t)} = g_o^{(t)} \odot \tanh(g_c^{(t)}) \tag{3.6.6}$$

$$y^{(t)} = \sigma\left(W^{og_o}s^{(t)}\right) \tag{3.6.7}$$

where W terms denote the weight matrices (e.g. W^{xg_i}, is the weight matrix from the input layer to the input gate), $W^{Sg_i}, W^{Sg_f}, W^{Sg_o}$ are diagonal weight matrices for peephole connections, σ is the logistic sigmoid function, $g_i^{(t)}, g_f^{(t)}, g_i^{(o)}$, and $g_c^{(t)}$ are respectively the input gate, forget gate, output gate and cell activation. The symbol $\odot$ represent the element-wise product of vectors, *tanh*, σ are the cell input and cell output activation functions. The output vector $y^{(t)}$ is a combination of the output gate and the cell state which contains stored information.

The most important key to understand LSTM networks is to understand the activation functions. How these function can be able to capture important information from the incoming signal and to blocks store and forget information.
We now let H be the size of the hidden state of LSTM unit. This is also called the capacity of an LSTM and is chosen by a user depending upon the amount of input data available and capacity of LSTM required. We also let B be the size of the input batch. The batches of data are usually fed into any LSTM based modeling form of a subset of a total number of example. Below are details associated with each parameter in the network described by Equations (3.6.1), (3.6.2), (3.6.3), (3.6.4), (3.6.5), (3.6.6) and (3.6.7).

- $x^{(1)}, ..., x^{(t-1)}, x^{(t)}, x^{(t+1)}, ..., x^{(T)}$: the input price vector with total length T.
- $x^{(t-1)} \in \mathbb{R}^B$, input price vector at time $t-1$.
- $W^{xg_i}, W^{xg_f}, W^{xg_o}, W^{xg_c} \in \mathbb{R}^{H\times B}$: weight matrix used to condition the input price vector, $x^{(t)}$ for all gates described in Section 3.6.1.
- $W^{Sg_i}, W^{Sg_f}, W^{Sg_o}, W^{Sg_c} \in \mathbb{R}^{H\times H}$: weight matrix used the output to previous time-step, $s^{(t-1)}$.
- $s^{(t-1)} \in \mathbb{R}^H$; output of the non-linear function at the previous time-step, $t-1$. $s^{(0)} \in \mathbb{R}^H$ is an initialization vector for the hidden layer at time step $t = 0$.
- σ: non-linear function (sigmoid here).
- $y^{(t)} = \sigma\left(W^{og_o}s^{(t)}\right)$: the output over the price at each time-step t. Essentially, $y^{(t)}$ is the next predicted price given the output of the hidden layer $s^{(t-1)}$ and the last observed

price vector $x^{(t-1)}$. Here $W^{og_o} \in \mathbb{R}^{|V| \times H}$ and $y^{(t)} \in \mathbb{R}^{|V|}$ where $\mid V \mid$ is the size of the output vector.

We can gain an insight into the structure of an LSTM by considering its architecture as following stages:

- **New memory generation**: We denote the input price $x^{(t-1)}$ and the past hidden state $s^{(t-1)}$ to generate a new memory $g_c^{(t)}$ which includes aspects of a new price $x^{(t-1)}$.
- **Input Gate**: We see that the new memory generation phase does not check whether the new input price is important before the new memory is generated - this is exactly the input gate's function. The input gate uses the input price and the past hidden state to determine whether or not the input is worth preserving and thus is used to gate the new memory.
- **Forget Gate**: This gate is similar to the input gate except that it does not make a determination of usefulness of the input price, instead, it makes an assessment on whether the past memory cell is useful for the computation of the current memory cell. It looks at $s^{(t-1)}$ and $x^{(t-1)}$, and outputs a number between 0 and 1 for each number in the cell state and produces $g_f^{(t)}$.
- **Final memory generation**: This stage first takes the advice of the forget gate $g_f^{(t)}$ and accordingly forgets the past memory $g_c^{(t-1)}$. Similarly, it takes an advice of input gate $g_i^{(t)}$ and accordingly gates the new memory $\hat{g}_c^{(t)}$. It finally sums up these two results to produce final memory $g_c^{(t)}$.
- **Output/Exposure Gate**: The main purpose of this gate is to separate the final memory from the hidden state. We also note that the final memory contains information that is not necessarily required to be saved in the hidden state. This gate makes the assessment regarding what parts of the memory $g_c^{(t)}$ needs to be exposed/present in the hidden state $s^{(t)}$. The signal produced indicate this is $g_o^{(t)}$ and this is used to gate the pointwise tanh of the memory.

It is still beneficial to incorporate the past prediction as an additional input to the output layer. The model described by Equation (3.6.7) exploits LSTM output and perform regression on the predictions.

Chapter 4

Error Correction Neutral Network

In this chapter we introduce a new recurrent neural network architecture that includes an additional input to the previous model error. Hence the learning can be interpreted as the model misspecification as an external shock to guide the model dynamical process. This is the idea behind the Error Correction Neural Network (ECNN) which was developed by Zimmermann [70]. In most cases, dynamic systems have an autonomous and an external force part. Many times relevant external forces may be hard to identify and data could be complex, non-stationary and noisy. The ECNN aims to reduce the over-fitting problem caused by the forecasting financial markets.

Recurring connections in ECNN enable memory effects to be included. A representation of such a recursion in form of a neural network is given in Section 4.1. In this context, the major difficulty is the identification of such complex systems, which is the parameter optimization task. This technique is a complete opposite approach of multi-layered perceptron (see Section 2.4.1) with a pattern recognition approach, instead the error correction network views the forecasting problem as a system identification problem, with the task of finding the dynamic system that best explains the data. In addition, this error correction neural network serves as an indicator of the correctness of the model. The ECNN can therefore be used to guide dynamics and thereby prevent the autonomous part of the model from learning false temporal dependencies [70]. The ECNN architecture is unfolded through time to transform the temporal identification task into spatial architecture, which can be handled by the shared weights extension of back-propagation-through-time (see Section 3.3). In addition to this, the unfolding introduces a temporal structure into the network architecture. In Section 4.3, we also introduce the technique of overshooting as a means of forcing the network to focus on the autoregressive part of the data dynamics during the training [70].

A basic dynamical recurrent system depicted in Figure 4.1 and at time t can be expressed by Equations (4.0.1) and (4.0.2). Note that Equation (4.0.1) without external force $y^{(t-1)} - \hat{y}^{(t-1)}$ would represent an autonomous system. If we are unable to identify the hidden patterns of the underlying systems dynamics due to insufficient input information or unknown influences, we can refer to the observed model error at the previous time $t-1$, which act as an additional input to the system. We extended the RNN by adding an error term $y^{(t-1)} - \hat{y}^{(t-1)}$ as an input to get the following:

$$s^{(t)} = f\left(s^{(t-1)}, x^{(t-1)}, y^{(t-1)} - \hat{y}^{(t-1)})\right) \tag{4.0.1}$$

$$y^{(t)} = g(s^{(t)}). \tag{4.0.2}$$

where $\hat{y}^{(t-1)}$ is the observed data at previous time $t-1$, g and f are activation functions.

This is similar to the MA part of linear ARIMA models, which utilizes both, linear autoregressive components and stochastic moving average-components derived from the observed model error to fit a time series (see [62]). As a major difference, the error correction system is a state space model, which accumulate memory during recursive as the network unfolds through time. Since we are working with state space models we can skip the use of delayed error correction. Another difference is that ARIMA models include only linear components, whereas our approach is non-linear.

4.1 Overview of An Error Correction Neural Network

A first neural network mathematical description of error correction neural network represented by Equations (4.1.1) and (4.1.2) can be formulated as

$$s^{(t)} = \tanh\left(As^{(t-1)} + Bx^{(t-1)} + D\left(Cy^{(t-1)} - \hat{y}^{(t-1)}\right)\right) \tag{4.1.1}$$

$$y^{(t)} = Cs^{(t)} \tag{4.1.2}$$

where $\left\{A, B, C, C\right\}$ are set of weight parameters. The term $Cs^{(t-1)}$recomputes the last output $y^{(t-1)}$ and compares it to the observed data $\hat{y}^{(t-1)}$. Note that a numerical ambiguity arises. Both weight matrix A and DC could code autoregressive structure of the system. However, adding a non-linearity is a measure to avoid this problem [27]. This yields

$$s^{(t)} = \tanh\left(As^{(t-1)} + Bx^{(t-1)} + D\tanh\left(Cy^{(t-1)} - \hat{y}^{(t-1)}\right)\right) \tag{4.1.3}$$

Using the tanh(.) as a squashing activation function, it is well known, that the numerics works best if the included variable fluctuates around zero, thus fits best for finite state space $(-1;1)^n$ created by the tanh(.) non-linearity.

Below are details associated with each parameter in the network described by Equations (4.1.1) and (4.1.2).

- $x^{(1)}, ..., x^{(t-1)}, x^{(t)}, x^{(t+1)}, ..., x^{(T)}$: the input price vector with total length T.
- $x^{(t-1)} \in \mathbb{R}^d$, input vector (price here) at time $t-1$.
- $B \in \mathbb{R}^{D_s \times d}$: weight matrix used to condition the input vector (price here), $x^{(t)}$, where D_s is the size of the hidden state.
- $A \in \mathbb{R}^{D_s \times D_s}$: weight matrix used the output to previous time-step, $s^{(t-1)}$.
- $D \in \mathbb{R}^{D_s \times |V|}$: where $|V|$ is the size of the output vector.
- $s^{(t-1)} \in \mathbb{R}^{D_s}$, output of the non-linear function at the previous time-step, $t-1$, where $s^{(0)} \in \mathbb{R}^{D_s}$ is an initialization vector for the hidden layer at time step $t = 0$.
- σ: non-linear function (sigmoid here).
- $y^{(t)} = Cs^{(t)}$: the output over the price at each time-step t. Essentially, $y^{(t)}$ is the next predicted price given the output of the hidden layer $s^{(t-1)}$ and the last observed input vector $x^{(t-1)}$. Here $C \in \mathbb{R}^{|V| \times D_s}$ and $y^{(t)} \in \mathbb{R}^{|V|}$ where $|V|$ is the size of the output vector.

The system identification is a parameter optimization task adjusting the weight of the four matrices A, B, C and D, that is minimizing the loss, L, over A, B, C, and D as follow

$$\min_{A,B,C,D} L := \frac{1}{2T} \sum_{i=1}^{T} \left(y^{(t)} - \hat{y}^{(t)} \right) \tag{4.1.4}$$

The ECNN offers the forecast based on the modelling of the recursive structure (matrix A), the external force (matrix B) and the error correction part (matrices C and D).

4.2 Unfolding in Time of An Error Correction Neural Network

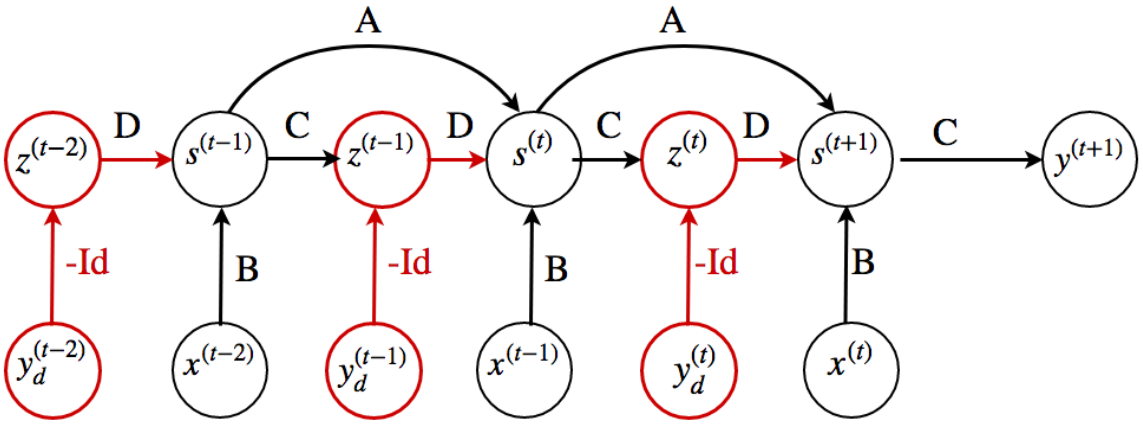


Figure 4.1: Unfolded error correction neural network where the weight matrices A, B, C, and D are shared weights and $-Id$ is the negative identity matrix. An ECNN whose only recurrence is the feedback connection from the error $z^{(t-2)}$ calculated from previous time step $t-2$ to output to the hidden layer. At each time step t, the input is $x^{(t)}$, the hidden layer activations are $s^{(t+1)}$, the outputs are $y^{(t+1)}$.

The unfolding of error correction neural network through time is described in this section. The resulting network architecture can be seen in Figure 4.1. In an error correction neural network, errors can be propagated further, i.e more than 2 layers, in order to capture longer historical information. This process is usually referred to as *unfolding*, an unfolding ECNN is shown in Figure 4.1. The ECNN architecture (Fig 4.1) is best to understand if one investigates the dependency between $s^{(t-1)}, x^{(t-1)}, z^{(t-1)}$ and $s^{(t)}$. The interesting part of ECNN is that it has two types of input:

(i) the external inputs $x^{(t)}$ directly influencing the state transition $s^{(t)}$.

(ii) the target, whereby only the difference between the internal prediction $y^{(t-1)}$ and the observation $\hat{y}^{(t-1)}$ has impact on the hidden state transition $s^{(t)}$.

Note that $-Id$ is the negative of the identity matrix. In addition, referencing from Figure 4.1, five different types of clusters can be observed; state cluster $s^{(t)}$, input cluster described by $x^{(t-1)}$, target clusters $\hat{y}^{(t-1)}$, error output cluster $z^{(t-1)}$ and output cluster $y^{(t)}$

4.3 Overshooting in Error Correction Neural Network

The dynamical system architecture represented by Figure 4.2 is the extention of the autonomous recurrence in future direction $t+1, t+2, ..., t+N$. Note that the overshooting

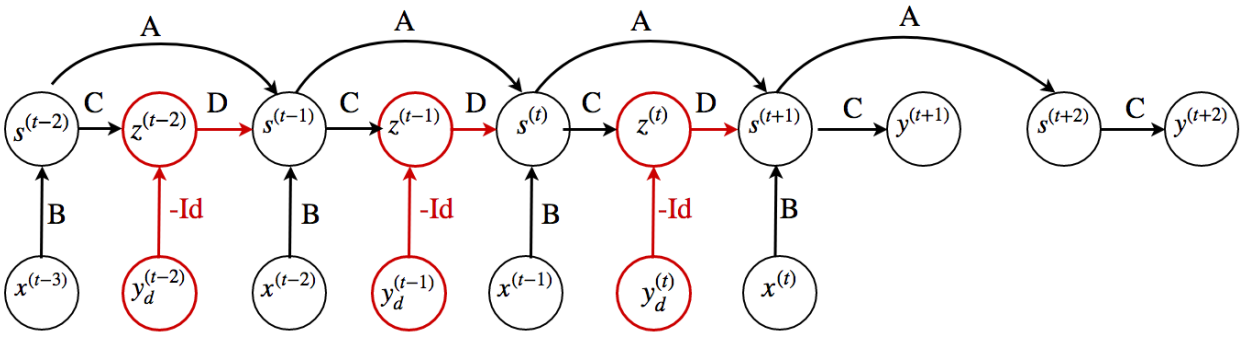


Figure 4.2: ECNN incorporating overshooting. Note that $-Id$ is the fixed negative of an identity matrix, while $z^{(t-1)}$ are outputs clusters to model the error correction mechanism (see comment in Figure 4.1).

part generate additional valuable forecast information about the dynamical system and act as a regularization method during learning.

4.4 Computing the Gradient in Error Correction Neural Network

The parameters of our ECNN are A, B, C and D, so we must compute the gradient of our loss function with respect to these matrices. We compute these gradients by using BPTT which is mentioned in details in Section 3.3. The procedural steps for implementation of the BPTT in an ECNN are given in the algorithm below:

(i) Forward pass: Step through $k^{(1)}$ time steps, compute the input, hidden, and output states.

(ii) Compute the loss, summed over the previous time steps.

(iii) Backward pass: Compute the gradient of the loss function w.r.t. to all the parameters A, B, C and D, accumulating over the previous $k^{(2)}$ time steps (this requires having stored all activations for time steps). In case of exploding and vanishing gradient we then clip gradients to avoid such problems.

(iv) Update parameters (this occurs once per batch of training data, not incrementally at each step).

(v) If processed multiple batches of a longer sequence, store the hidden state at the last time step (which is used to initialize state for beginning batch). If the end of the sequence data is reached, reset the memory (hidden state) and move to the beginning of the next sequence.

(vi) Repeat from step 1.

We need to compute the gradient of our loss function with respect to the weight matrices $A, B.C, D$. The parameter C is present in two functions which are *state transition* with the *external force* and *output equation* as shown by Equation (4.1.3). Lets consider the network parameters in Figure 4.1 as the set $\beta = \{A, B, C, D\}$, and $s^{(t)}$ as the hidden state of the network at time t, we can write the gradients as

$$\frac{\partial L}{\partial \beta} = \sum_{t=1}^{T} \frac{\partial L^{(t)}}{\partial \beta} \tag{4.4.1}$$

where $L^{(t)}$ is the loss described in Section 3.3 by Equation (3.3.5) the expansion of loss function gradients at time t is

$$\frac{\partial L^{(t)}}{\partial \beta} = \sum_{k=1}^{t} \frac{\partial L^{(t)}}{\partial y^{(t)}} \frac{\partial y^{(t)}}{\partial s^{(t)}} \frac{\partial s^{(t)}}{\partial s^{(k)}} \frac{\partial s^{(k)}}{\partial \beta}. \tag{4.4.2}$$

Our loss function is described in Section 3.3 by Equation (3.3.5). It describes how parameters in the set β affect the loss function at the previous time-steps (i.e., $k < t$).

To gain some intuition for how the BPTT algorithm behaves, we provide an example of how to compute gradients by BPTT for the ECNN equations above (Equation (4.1.1) and Equation (4.1.2)). The nodes of our computational graph include the parameters A, B, C and D as well as the sequence of nodes indexed by t for $x^{(t)}$, $s^{(t)}$, $\hat{y}^t$, and $y^{(t)}$. The gradient $\nabla_{y^{(t)}} L$ on the outputs at time step t, for all i, t, is as follows:

$$\left(\nabla_{y^{(t)}} L\right)_i = \frac{\partial L^{(t)}}{\partial y_i^{(t)}} = y_i^{(t)} - \hat{y}_i^{(t)} \tag{4.4.3}$$

if we work our way backward, starting from the end of the sequence. At the final time step T, $s^{(T)}$ only has $y^{(T)}$ as a descendent, so its gradient is simple:

$$\nabla_{s^{(T)}} L = \left(\nabla_{s^{(T)}} y^{(T)}\right)^{\top} \nabla_{y^{(T)}} L = C^{\top} \nabla_{y^{(T)}} L \tag{4.4.4}$$

where $\nabla_{y^{(T)}} L = y^{(T)} - \hat{y}^{(T)}$, $\hat{y}^{(T)}$ and $y^{(T)}$ are the target and predicted values at the final time step T respectively. For each time $t < T$, in decreasing order of t, we proceed backward along each path starting at $s^{(t+1)}$ and finishing at $s^{(t)}$ and also starting at the $L^{(T)}$ proceed backward similarly until reaching $s^{(t)}$. Adding up the derivatives (along three paths here) and set the sum equal to $\frac{\partial L}{\partial s^{(t)}}$, the following expression is obtained:

$$\nabla_{s^{(t)}} L = \left(\frac{\partial s^{(t+1)}}{\partial s^{(t)}}\right)^{\top} \left(\nabla_{s^{(t+1)}} L\right) + \left(\frac{\partial y^{(t)}}{\partial s^{(t)}}\right)^{\top} \left(\nabla_{y^{(t)}} L\right). \tag{4.4.5}$$

Now at time step $t+1$ Equation (4.1.3) becomes

$$s^{(t+1)} = \tanh\left(As^{(t)} + Bx^{(t)} + D \cdot \tanh\left(Cs^{(t)} - y_d^{(t)}\right)\right). \tag{4.4.6}$$

Equations (4.1.3) and (4.4.6) enables us to obtain our Jacobian matrix for the hidden state parameter as

$$\begin{aligned}\left(\frac{\partial s^{(t+1)}}{\partial s^{(t)}}\right)^{\top} &= \left(A^{\top} + \left[\frac{\partial}{\partial s^{(t)}}\tanh(Cs^{(t)} - \hat{y}^{(t)})\right]^{\top} \cdot D^{\top}\right)\operatorname{diag}\left(1 - \left(s^{(t+1)}\right)^2\right), \\ &= \left(A^{\top} + \left[\operatorname{diag}\left(1 - \left(z^{(t)}\right)^2\right) \cdot C\right]^{\top} \cdot D^{\top}\right)\operatorname{diag}\left(1 - \left(s^{(t+1)}\right)^2\right), \\ &= \left(A^{\top} + C^{\top} \cdot \operatorname{diag}\left(1 - \left(z^{(t)}\right)^2\right) \cdot D^{\top}\right)\operatorname{diag}\left(1 - \left(s^{(t+1)}\right)^2\right)\end{aligned} \tag{4.4.7}$$

Also $\left(\frac{\partial y^{(t)}}{\partial s^{(t)}}\right)^{\top}$ is given by

$$\left(\frac{\partial y^{(t)}}{\partial s^{(t)}}\right)^{\top} = C^{\top} \tag{4.4.8}$$

we then insert Equation (4.4.7) and (4.4.8) into Equation (4.4.5) to get

$$\begin{aligned}\nabla_{s^{(t)}} L = &\left(A^{\top} + C^{\top} \cdot \operatorname{diag}\left(1 - \left(z^{(t)}\right)^2\right) \cdot D^{\top}\right)\operatorname{diag}\left(1 - \left(s^{(t+1)}\right)^2\right)\left(\nabla_{s^{(t+1)}} L\right) \\ &+ C^{\top}\left(\nabla_{s^{(t+1)}} L\right)\end{aligned} \tag{4.4.9}$$

where $z^{(t)}$ represent input error $\left(Cs^{(t)} - \hat{y}^{(t)}\right)$ and $\operatorname{diag}\left(1 - (s^{(t+1)})\right)$, $\operatorname{diag}\left(1 - (z^{(t)})\right)$ indicates the diagonal matrix containing the elements $1 - \left(s^{(t+1)}\right)^2$ and $1 - (z^{(t)})^2$ in the main diagonal. Once the gradients on the internal nodes of the ECNN are obtained, we can obtain the gradients on the parameter nodes.

Note that parameter A appears in the arguments for both $s^{(t)}$ and $y^{(t)}$, so we will have to check in both $s^{(t)}$ and $y^{(t)}$. We also make note that $y^{(t)}$ depends on A both directly and indirectly (through $s^{(t-1)}$). Using notation described in Section 3.3, the gradient on the remaining parameters is given by:

$$\nabla_A L = \sum_t \sum_i \left(\frac{\partial L}{\partial s_i^{(t)}}\right) \nabla_A s_i^{(t)} \tag{4.4.10}$$

The final term, however, requires us to notice that there is an implicit dependence of $s^{(t)}$ on A_{ij} through $s^{(t-1)}$ as well as a direct dependence. Hence, we have the expression described by Equation (7.2.1) that's for $\frac{\partial L}{\partial s^{(t)}}$. Now, the derivative of the internal hidden state with respect to weight matrix A

$$\frac{\partial s^{(t)}}{\partial A} = \operatorname{diag}\left(1 - \left(s^{(t)}\right)^2\right) s^{(t-1)\top} \tag{4.4.11}$$

we then insert Equations (4.4.11) into Equation (4.4.10) to get

$$\nabla_A L = \sum_{t=1}^{T}\left(\operatorname{diag}\left(1 - \left(s^{(t)}\right)^2\right)\left(\nabla_{s^{(t)}} L\right) s^{(t-1)\top}\right) \tag{4.4.12}$$

where $\nabla_{s^{(t)}} L$ is described by Equation (4.4.5). Taking the gradient of B and D is similar to doing it for A since they both require taking sequential derivatives of an $s^{(t)}$ vector. We get the following expressions

$$\nabla_B L = \sum_{t=1}^{T} \left(\text{diag}\left(1 - \left(s^{(t+1)}\right)^2\right) (\nabla_{s^{(t)}} L) x^{(t-1)\top} \right) \tag{4.4.13}$$

$$\nabla_D L = \sum_{t=1}^{T} \left(\text{diag}\left(1 - \left(s^{(t+1)}\right)^2\right) (\nabla_{s^{(t)}} L) z^{(t-1)\top} \right) \tag{4.4.14}$$

In order to find the derivative of the loss function with respect to parameter C, we firstly define $\nabla_C L$ as

$$\nabla_C L = \sum_t \sum_i \left(\nabla_{y_i^{(t)}} L \right) \left(\nabla_C y_i^{(t)} \right) \tag{4.4.15}$$

Since $y^{(t)} = Cs^{(t)}$, then Equation (4.4.15) becomes

$$\begin{aligned} \nabla_C L &= \sum_t \sum_i \left(\nabla_{y_i^{(t)}} L \right) \nabla_C (Cs^{(t)})_i \\ &= \sum_t \left(\nabla_{y^{(t)}} L \right) s^{(t)\top} + \sum_t \sum_i \sum_j C_{ij} \left(\nabla_{y_i^{(t)}} L \right) \left(\nabla_C s^{(t)} \right)_j \\ &= \sum_t \text{diag}\left(1 - \left(z^{(t-1)}\right)^2\right) \cdot D^\top \text{diag}\left(1 - \left(s^{(t)}\right)^2\right) \cdot C^\top \left(\nabla_{y^{(t)}} L \right) s^{(t-1)\top} \\ &+ \sum_t \left(\nabla_{y^{(t)}} L \right) s^{(t)\top}. \end{aligned} \tag{4.4.16}$$

We finally apply the optimization technique called *Adagrad* which is explained in details in Section 3.5.2, we update our parameters as :

$$\beta^{(t+1,i)} = \beta^{(t,i)} - \frac{\eta}{\sqrt{G^{(t,ii)} + \epsilon}} \cdot g^{(t,i)} \tag{4.4.17}$$

where $\beta = \left\{ A, B, C, C \right\}$ (update each set of parameter), $G^{(t)}$ is a diagonal matrix where each diagonal element i, i is the sum of the square of the gradients w.r.t β_i up to time step t [56], while ϵ is a smoothing term that avoids division by zero (usually set to an order of 10^{-8}).

4.5 Extension of Error Correction Neural Network

There exist several extensions of error correction neural network, a few are stipulated in the list below. For more extensive information see [71].

4.5.1 Variants-Invariants Separation

Forecasting a high dimensional dynamical system (i.e. several target series) is difficult. A way of reducing the complexity of the task is to separate the dynamics into time variant and invariant structure. In most cases, one lets the system forecast the variants and eventually merge this forecast with the unchanged invariant. This can be done by recombining the

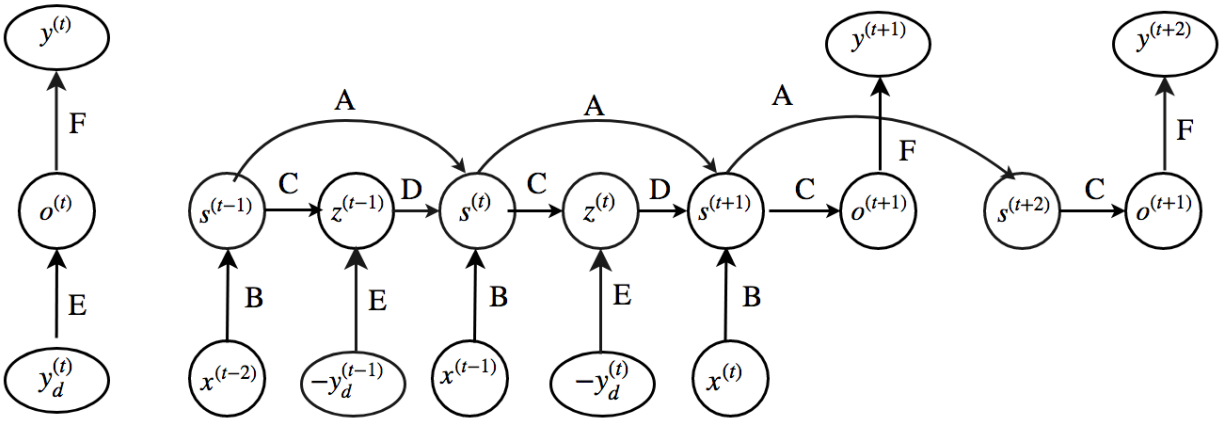


Figure 4.3: ECNN with Variant-Invariant Separation (see comments in Figure 4.1).

standard ECNN (Figure 4.2) to a compression-decompression network shown in Figure 4.4, this transformation reduces the dimensionality of the original forecast problem.
The compressor E separates the time variants from invariants structures while matrix F reconstruct the dynamics. By such transformation, the ECNN has to predict a coordinate transformation low dimensional vector $o^{(t)}$ instead of the high dimensional vector $y^{(t)}$ [73]. It is important to note, that ECNN require $-y_d^{(t)}$ (an observed value at time t) as input in order to generate $o^{(t)}$ in $z^{(t)}$ layer. This enables the compensation of the internal state forecast $o^{(t)} = Cs^{(t)}$ by the transformed observation $-o_d^{(t)} = E(-y_d^{(t)})$. The set of weight matrices $\left\{B, E, F, D, C\right\}$ used in ECNN with *variant-invariant separation* are share throughout time, the result of the bottleneck network is also transferred to the ECNN branch (right-hand-side of Figure 4.3), refer to [73] for a detailed description. One of the successful application of the *variant-invariant* separation is reported in Zimmermann [71]. The authors for this paper use the extension of ECNN depicted by Figure 4.3 to forecast the complete German yield curve that contains ten different interest rates maturities. In a benchmark comparison on a common performance matric, it turns out that the extended ECNN is able to outperform the benchmarks (naive strategy see [45]) on short and long-term (e.g monthly, quarterly and semi-annual) predictions.

4.5.2 State Space Reconstruction

The aim of this extension of the error correction neural network is to force the state transitions such that the related forecast problem becomes easier because the transformed system evolves more smoothly over time. This is achieved by integrating state space reconstruction and forecasting in a neural network approach. In addition to a more smooth transition of the states, there also is an improvement of velocity control and noise reduction [67] (see Figure 4.4).

4.5.3 Undershooting

Undershooting is the refinement of the model time grid by recurrent neural networks, which is directly related to the considerations on uniform causality (provides the basis for the embedding of a discrete time systems) [72]. Time series analysis comprises with building a model of a dynamical system on the basis of observed data [69]. The time grid of the data is identically

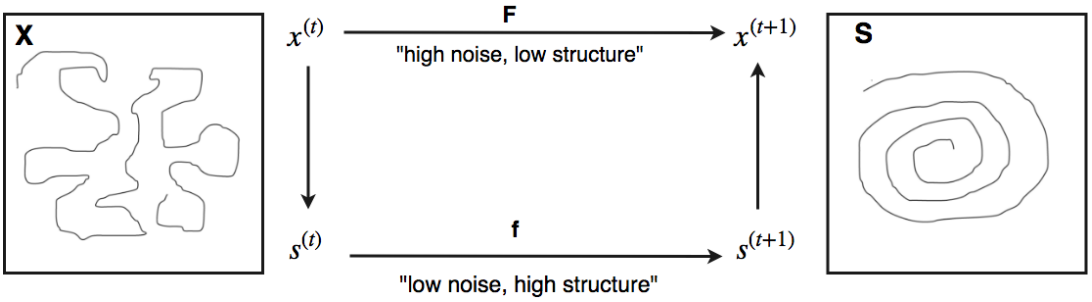


Figure 4.4: The time series of observed state description $x^{(t)}$ may follow a very complex trajectory. A transformation to a possible higher-dimensional state space may result in a smother trajectory, adopted from [68].

to that of a discretized time grid of the model. Using undershooting, the system dynamics to be identified by the ECNN can be better understood [67].

Chapter 5

Empirical Study

This chapter described the procedures used to obtain the results. Section 5.1 describes the selection of stocks, as well as how stocks are pre-processed. Section 5.1.3 states the problem definition and how the models are used to address the main object of this thesis. We further address the experimental design in Section 5.2 were we discuss the prediction approach, training and performance evaluation. Finally, Sections 5.2.3 and 5.2.4 describe the performance measures and trading strategy respectively.

5.1 Stock Data

In this section, we provide details of our selection of samples and the input variables we choose for model prediction and the data resources is provided. As a representation of each market, a stock market index was used. The four stock indices we choose are S&P 500, Hang Seng index, Nikkei 225 and DJIA index. These indices can be seen as an average of the most impactful stocks on a certain market as discussed in this section. This is a weighted average so that a large company (such as Apple and Microsoft in the USA) has a greater impact on the index value (such as S&P 500).

For each market in Table 5.1 standard RNN, LSTM, and ECNN models were trained over the period from 2002-01-01 to 2016-01-01, using historical stock data from Yahoo Finance [35]. For each database entry of the stock index listed in Table 5.1 includes, information concerning closing, opening, high and low prices and volume of trades when applicable. The selected period of data equality has also been examined, including ensuring that the inequality $high \geq close \geq low$ is met when possible.

Symbols	Name	Country
GSPC	S&P 500	USA
HSI	Hang Seng Index	Hong Kong
DJIA	Dow Jones Industrial Average	USA
N225	Nikkei 225	Japan

Table 5.1: The stock selection

5.1.1 Data Section Descriptions

S&P 500 is an equity index that includes 500 leading companies in the US economy's leading industries [36], claims that the index covers approximately 80% percent of the market capitalization available, making it a relevant research topic. The S&P 500 price development is shown in Figure 5.1.

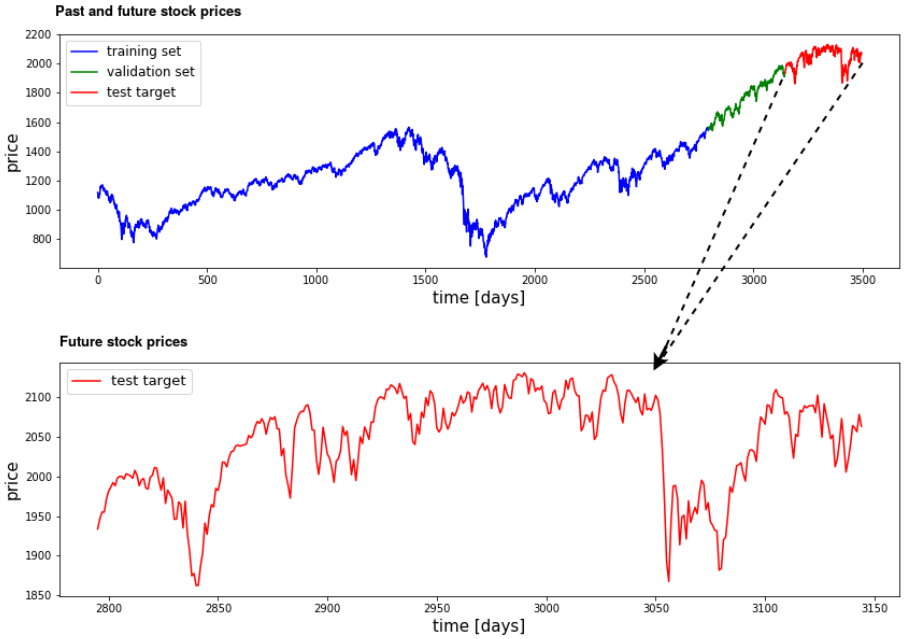


Figure 5.1: GSPC daily closing price development between the period of 2002-01-01 and 2015-12-30.

HSI is the market-weighted index of the largest trading companies on the Hong Kong Exchange. The Hong Kong stock exchange accounts for 67% of capitalization. The components of the index are divided into four subindices: Commerce and Industry, Finance, Utilities, and Properties. HSI price development is shown in Figure 5.2.

DJIA is the benchmark stock market index (the Dow Jones Industrial Average) is the price-weighted average of 30 major stocks traded on the New York Stock Exchange (NYSE) and the Nasdaq stocks. The DJIA historical price development is shown in Figure 5.3.

N225 is the largest and most respected Japanese stock index. It is a price- weighted index made up of the Tokyo Stock Exchange top 225 blue- chip companies. This is an adjusted price index that maintains its continuity through the "Dow" method and rules for integrating Japanese specific trading practices. Figure 5.4 illustrates the daily closing price movement of N225.

The liquidity of the market and the sector balance periodically reviews the N225 constituent stocks. The index, calculated with highly liquid stocks, aims to achieve two objectives, one is to maintain its long-term continuity and the other to reflect changes in the structure of the industry.

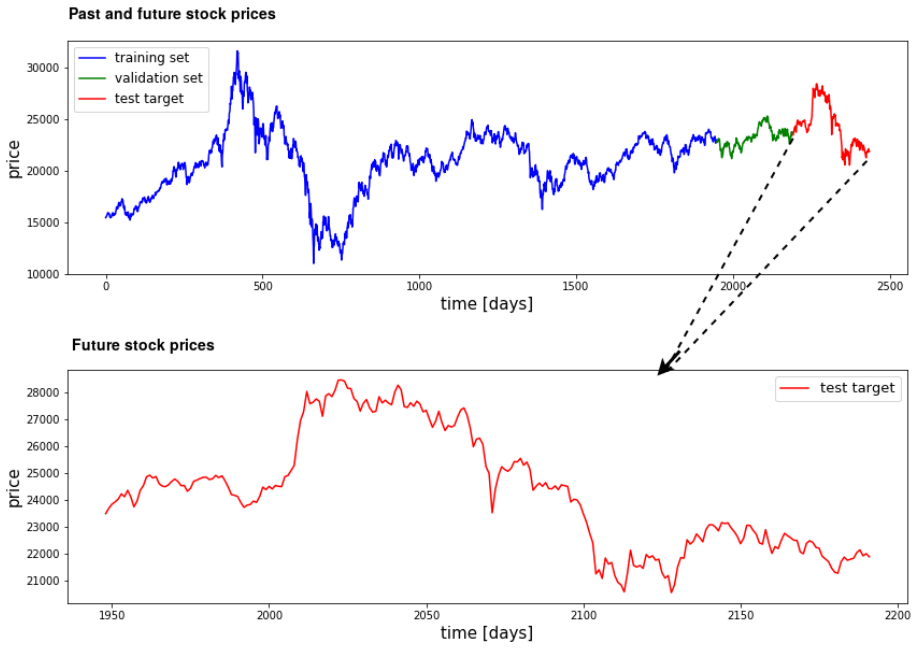


Figure 5.2: HSI daily closing price development between the period of 2002-01-01 and 2015-12-30

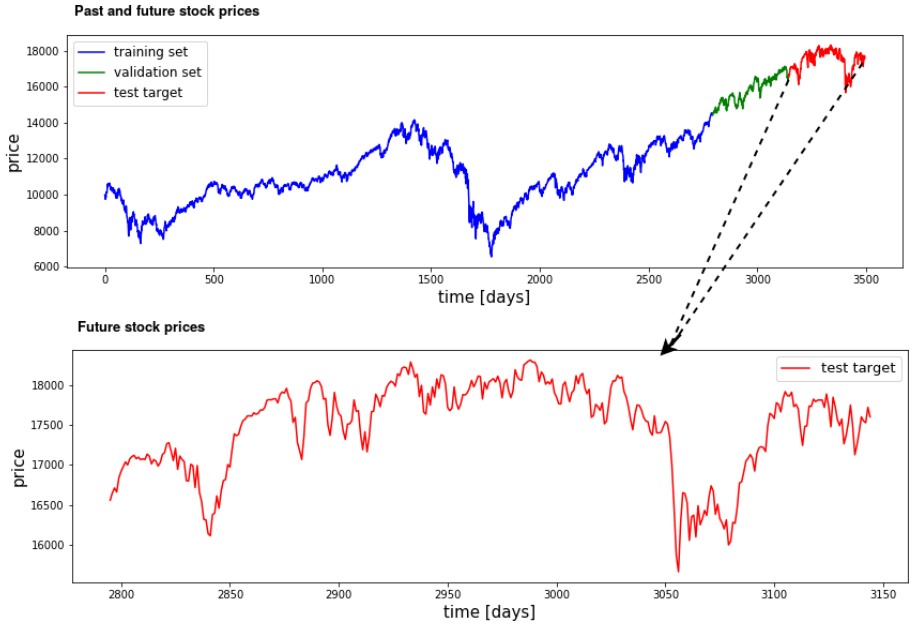


Figure 5.3: DJIA daily closing price development between the period of 2002-01-01 to 2015-12-30.

5.1.2 Model Input

In a survey of forecasting approaches, Atsalakis and Valvanis [5] indicated that technical analysts typically use indicators to forecast future prices. Four types of variables are used as model inputs for each stock index. The first set consists of historical stock trading data such as the Open, High, Low and Close Price (OHLC) (see Section 2.1.2) [15, 50], and the second is the technical indicators of a stock trade. These are commonly used inputs in previous studies [66]. Technical analysts usually use indicators to predict the future. The major types of indicators are *Moving Average (MA)*, *Exponential Moving Average (EMA)*, *Moving Average Con-*

Figure 5.4: Nikkei 225 daily closing price development between the period of 2002-01-01 to 2015-12-30.

vergence Divergence (MACD), *Average True Range (ATR)* and *Stochastic (% K)*. The mathematical description for the selected technical indicators are describes bellow:

Moving Average

A moving average (MA) is a trend indicator that dynamically calculates the mean average of prices over a defined number of past. The MA is defined by the following expression:

$$M = CCP - OCP \tag{5.1.1}$$

where CCP and OCP are current closing and old closing price for a predetermined period (5 and 10 days).

Exponential Moving Average

Exponential Moving Average (EMA) for a predetermined period (20 days) in each point is calculated according to the following formula:

$$EMA^{(t)} = EMA^{(t-1)} + \alpha\left(x^{(t)} - EMA^{(t-1)}\right) \tag{5.1.2}$$

where

- $\alpha = \frac{2}{n+1}$
- n = the length of the EMA
- $x^{(t)}$ = the current closing price
- $EMA^{(t-1)}$ = the previous EMA value
- $EMA^{(t)}$ = the current EMA value

An exponential moving average (EMA), sometimes referred to as an exponentially weighted moving average (EWMA), applies weighting factors that exponentially decrease. The weight of each older data point decreases exponentially, giving much more importance to recent observations, although older observations are still not completely discarded.

Moving Average Convergence Divergence

Technical analysis is one of the method which provides the basis for decision-making in equity trading, one of the variables of the technical analysis is Moving Average Convergence Divergence (MACD) for a predetermined period (12 and 26 days). The MACD is constructed based on exponential moving averages. It is calculated by subtracting the longer exponential moving average (EMA) of window length N from the shorter EMA of window length M, where the EMA is computed as follows:

$$EMA^{(t)}(N) = \left(\frac{2}{N} \times \left(P^{(t)} - EMA^{(t-1)}(N)\right)\right) + EMA^{(t-1)}(N) \tag{5.1.3}$$

where $EMA^{(t)}(N)$ is the exponential moving average at time t, N is the window length of the EMA, and $P^{(t)}$ is the value of index at time t.

Average True Range

The average true range (ATR) inferred to as measures price volatility. Like any other average, ATR is calculated by selecting a period of 14 days, which is commonly used and by adding the true range of the present day to that of the previous 13 days, then dividing by 14. This would be your initial ATR:

$$\text{ATR} = (13 \text{ previous ATR} + \text{today's true range})/14 \tag{5.1.4}$$

Stochastic Fast (%K)

The stochastic fast $(\%K)$ is a momentum indicator comparing the closing price of a security to the range of its prices over a certain period of time and is define by the following expression:

$$\%K = \frac{CCP - L14}{H14 - L14} \tag{5.1.5}$$

where $L14$ and $H14$ denote the lowest low and highest high of the past 14 days respectively.

5.1.3 Problem Definition

We defined input with 10 days historical prices $x^{(i)} = \left\{x^{(t-9,i)}, x^{(t-8,i)}, ..., x^{(t,i)}\right\}$. The output value is defined by the next closing price $y^{(t+1,i)} \in \mathbb{R}$. As a more specific example, Fig 5.6 shows the relationship between the input values and the output value for stock i from a set of training data at 11$^{\text{th}}$ January 2002 at $t+1$. The period for the input data constituents 10$^{\text{th}}$ January 2002 (t), 9$^{\text{th}}$ January 2002 (*t-1*),..., 1$^{\text{st}}$ January 2002 (*t-9*) as factors of the ten points time. The output value is the stock closing price on 11$^{\text{th}}$ January 2002 (t+1). In order to reduce the impact of noise on the financial market and ultimately lead to a better prediction, the stock data collected above should be properly normalized. Different methods of normalization are tested to improve predictive performance during training [16], [44]. Which includes a

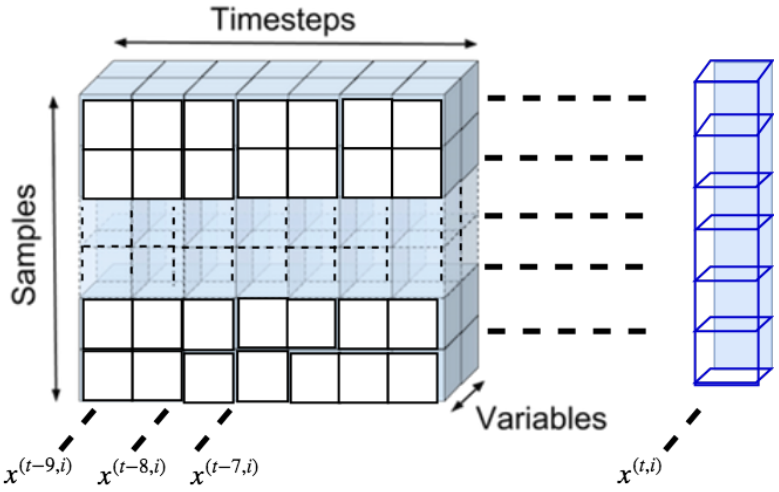


Figure 5.5: The is the data representational structure which has the shape (number of samples, rolling window, number of features)

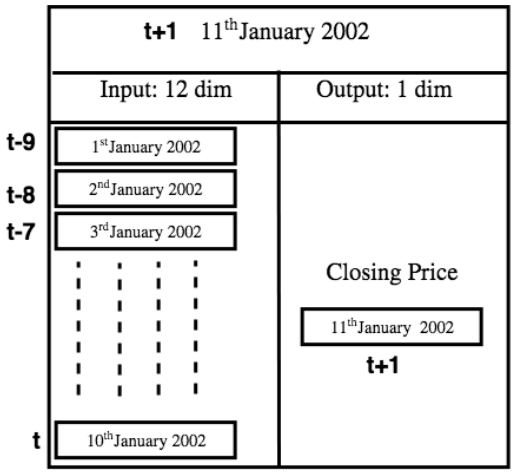


Figure 5.6: Stock i from one set of training data from 1th January 2002 to 10th

rescaling of data so that each input value is maximally 1 (minimum ≈ 0), this is often referred to as linear scaling using the minimum and maximum time series value to be scaled. Equation (5.1.6) scales the time series to the range [0,1] [52].

$$\hat{x}^{(t)} = \frac{x^{(t)} - \min(x^{(t)})}{\max(x^{(t)}) - \min(x^{(t)})} \tag{5.1.6}$$

where:

- $\hat{x}^{(t)}$ - Processed data value
- $x^{(t)}$ - Unprocessed data value
- $\min(x^{(t)})$ - Minimum value
- $\max(x^{(t)})$ - Maximum value

Similarly rescaling is done for output values $y^{(t+1,i)}$, to convert to the cross-sectional stock prices. Note that $\hat{y}^{(t+1,i)}$ is the observed values after data processing. In order to obtain the true value after forecasting, we can revert the output variables as

$$x^{(t)} = \hat{x}^{(t)}\Big(\max(x^{(t)}) - \min(x^{(t)})\Big) + \min(x^{(t)}) \tag{5.1.7}$$

This procedure is extended to using the latest N days rather than the most recent set of training data (one training data). We use the mean squared error (MSE) as the loss function and define $\text{MSE}^{(t+1)}$ when training the model at $t+1$ as follows:

$$\mathbf{MSE}^{(t+1)} = \frac{1}{K}\left\{\sum_{t=T-N+1}^{T}\sum_{i\in U_t}\left(\hat{y}^{(t,i)} - f\left(\hat{x}^{(t,i)}; W^{(t+1)}\right)\right)^2\right\} \tag{5.1.8}$$

In Equation (5.1.8), K is the number of all training examples, U_t is the input data universe at t. $W^{(t+1)}$ is weight matrix parameter calculated by solving Equation (5.1.8) and makes the form of activation function described by $f(.)$.

5.2 Experimental Design

We present details of how we achieve the predicted value and how we assess the performance of each model.

5.2.1 Prediction Approach

The prediction process follows Chan's prediction method [49]. This procedure consists of three parts. The first part is the training step to train the model and update the parameter of the model. The second part is the validation part. We use it to adjust hyperparameters and get an optimal model setting. The last part is the test part, in which we use the best model to predict data. In the training part, we use the past seven years' and eight month worth of daily stock data and selected indicated (see Section 5.1.2) to train the model. The following eleven months is employed to the validation part. In the test part, we predict a daily performance of each model. This process continues for six years each day from 19$^{\text{th}}$ March 2010 to 30$^{\text{th}}$ December 2015 as you can see on the top right side of Figure 5.7. Finally, for each stock index, there are approximately 336 days (almost 11 months) daily predicted results on the testing part. The first prediction for the six-year period was trained from 2002-04-19 to 2009-04-19 (7 years, 4 month) and also trained on the validation set from 2009-04-20 to 2010-03-19, then tested on the testing set from 2010-03-20. For every 336 predicted days, we extract the first 252 days which is equivalent to normal trading days a year. The prediction procedure is illustrated in Figure 5.7.

5.2.2 Training and Performance Evaluation

Most recent data were selected for testing, next recent dataset was selected for the validation and rest were selected for the training. The size of the test set was set to 10% of the whole dataset and the size of the validation set is also 10% of the whole data set. The rest (80%) of the data set was used to train the models.

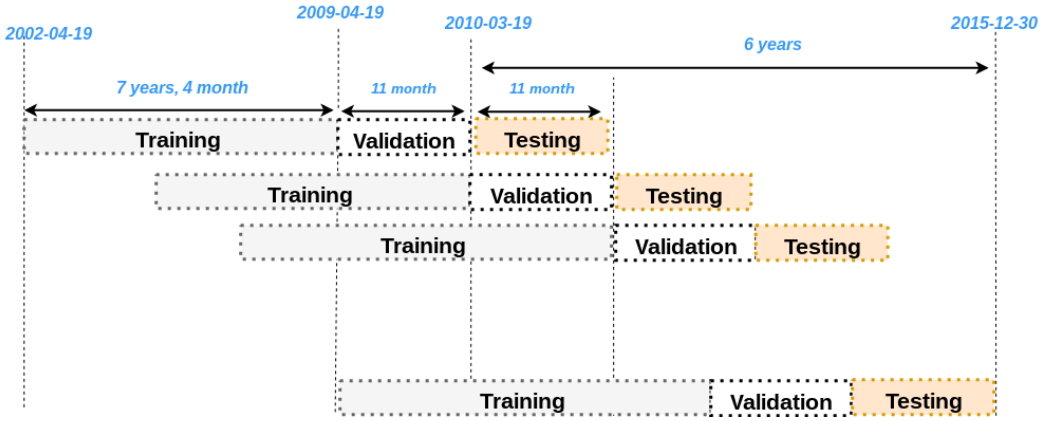


Figure 5.7: Continuous data set arrangements during the entire sample period for training, validation and testing.

5.2.3 Performance Criteria

Previous papers have selected indicators to assess how well the model predicts financial market trends [4, 18, 28, 33]. In this paper we follow their methods. However, we mainly focus on three classical indicators, Relative measure (Theil U), Mean Absolute Percentage Error (MAPE) and Pearson's Correlation Coefficient (R)) for measuring the predictive accuracy of each model. The mathematical definition of these indicators are discussed below

Theil U

Theil U is the relative measure of the difference between two variables. It squares the deviation to give more weight to large errors and exaggerates errors. Equation (5.2.1) defines Theil U precisely:

$$Theil\ U = \frac{\sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(\hat{y}^{(t)} - y^{(t)}\right)^2}}{\sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(\hat{y}^{(t)}\right)^2} + \sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(y^{(t)}\right)^2}} \tag{5.2.1}$$

where N is the number of predictions and $\hat{y}^{(t)}$ and $y^{(t)}$ are predicted and actual values, respectively.

Mean Absolute Percentage Error

The *Mean Absolute Percentage Error* $(MAPE)$ (MAPE) is used as a predictive accuracy measure to determine which model performs the best [44]. MAPE can evaluate and compare models' predictive power. The definition of MAPE is

$$MAPE = \frac{1}{N}\left(\sum_{t=1}^{N}\left|\frac{\hat{y}^{(t)} - y^{(t)}}{y^{(t)}}\right|\right) \tag{5.2.2}$$

where $\hat{y}^{(t)}$ is the target value and $y^{(t)}$ is the predicted value. A lower MAPE value indicates better network performance, but we cannot expect it to be close to zero, as financial markets

are so volatile and so fluctuating. The accuracy of the offered models will be comparable across the four futures markets even with their different scale of indices.

Pearson's correlation coefficient

Pearson's correlation coefficient (R) is the measurement of the linear correlation between two variables. Large R values mean more shared variation which means more accurate predictions are possible about one variable based on nothing more than knowledge of the other variable. Equation (5.2.3) defines R precisely:

$$R = \frac{\sum_{t=1}^{N}\left(y^{(t)} - \overline{y^{(t)}}\right)\left(\hat{y}^{(t)} - \overline{\hat{y}^{(t)}}\right)}{\sqrt{\sum_{t=1}^{N}\left(y^{(t)} - \overline{y^{(t)}}\right)^2\left(\hat{y}^{(t)} - \overline{\hat{y}^{(t)}}\right)^2}} \tag{5.2.3}$$

where y_t, $\hat{y}_t$ are actual and predicted values respectively and N represents the prediction period.

5.2.4 Trading Strategy

The direction of a price development is relevant to traders, as the direction directly may determine whether to take a long or a short position in the market. In this section, we address one of the trading strategies which aims to validate the direction of a price based on the future predictions and past actual values. A trading strategy is a way to buy and sell in markets based on predefined rules for trading decisions. Trading strategies are employed to avoid behavioral finance biases and ensure consistent results. In our case, we consider the strategy of buying and selling based on the predicted results of each model. The strategy recommends that investors buy when the next period's expected value is higher than the actual value. On the contrary, it recommends that investors sell when the predicted value at day t is smaller than the predicted value at day $t+1$. The strategy can be described by the following equations:

$$y^{(t+1)} > y^{(t)} \quad : \quad \text{Buy} \tag{5.2.4}$$

$$y^{(t+1)} < y^{(t)} \quad : \quad \text{Sell} \tag{5.2.5}$$

where $y^{(t)}$ is the current predicted closing price and $y^{(t+1)}$ is the predicted closing price for the following market day.

Directional Accuracy

$$DA = \frac{1}{m-1}\sum_{t=1}^{m-1} pos\left(\left(\hat{y}^{(t+1)} - \hat{y}^{(t)}\right)\left(y^{(t+1)} - y^{(t)}\right)\right) \tag{5.2.6}$$

where *pos* is an unary operator defined as:

$$pos = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if otherwise.} \end{cases} \tag{5.2.7}$$

This measures the proportion of the days when the model forecasts the correct direction of price movement.

Strategy Returns

The strategy described in this section is widely used for profitability performance. The total return of this trade rule can be calculated using the following equation and used as a scale of comparison between models and markets:

$$\hat{R} = 100 \times \left(\sum_{t=1}^{b} \frac{y^{(t+1)} - y^{(t)} + (y^{(t)} * B + y^{(t+1)} * S)}{y^{(t)}} + \sum_{t=1}^{s} \frac{y^{(t)} - y^{(t+1)} + (y^{(t+1)} * B + y^{(t)} * S)}{y^{(t)}} \right) \tag{5.2.8}$$

where R is the strategy returns, b and s denote the total number of days for buying and selling, respectively [6, 65]. B and S are the transaction costs for buying and selling, respectively. We choose the unified cost in the spot as 0.25% for buying and 0.45% for selling as used by Wei, Jun & Yulei paper [6].

Chapter 6

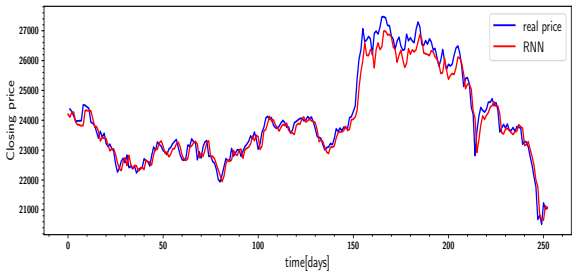
Results

In this chapter, the proposed and conventional forecasting approaches (RNN, LSTM and ECNN model) are compared using the four indices mentioned in Section 5.1.2. We set the same network inputs for these three different models, including four types of series: daily open price, daily closing price, daily highest price, daily lowest price, and technical indicators discussed in Section 5.1.2. A rolling window (rw) of a fixed number of days is taken and the closing price of the next day after the end of the window is predicted. For each stock index, the yearly predicted data from the three models and the corresponding real price are shown in graphical form. Figures 6.1a-6.1f, 6.4a-6.4f, 6.7a-6.7f and 6.10c-6.10f illustrate yearly predicted results of the HSI, NIKKEI 255 (N225), DJIA and S&P 500 indices for the three proposed models using the rolling window of 10 and 60. Figures 6.1c-6.1f, 6.4c-6.4f, 6.7c-6.7f and 6.10c-6.10f show that for each simulated stock index, RNN and LSTM have greater variations and distances to the real price than ECNN. Furthermore, loss function (MSE) represented by Figures 6.3a-6.3f, 6.6a-6.6f, 6.9a-6.9f and 6.12a-6.12f clearly show that ECNN has the least value of the loss as compared to RNN and LSTM in all the predicted stock indices.

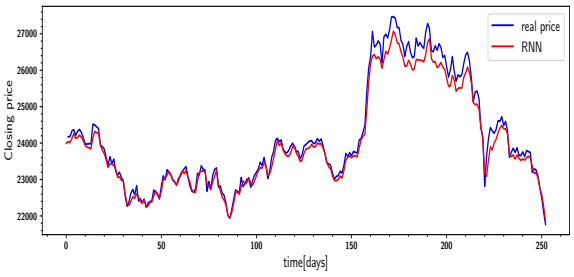
For Figures 6.1b, 6.1d, and 6.1f are simulated results by RNN, LSTM, and ECNN respectively, the rw is fixed at 60, with an increase in number of hidden state size. We can observe that an increase in rw and h_s result to an improvement of the prediction performance of each model, this is also true to other stock indices (see Figures 6.7b, 6.7d, 6.7f, 6.10b, 6.10d and 6.10f). In Figure 6.1a-6.1d, 6.4a-6.4d, 6.7a-6.7d and 6.10a-6.10d RNN and LSTM failed to identify the pattern at the beginning and also at some regions towards the end of the yearly traded days, whereas the prediction performance of the ECNN model represented by Figure 6.1e, 6.1f, 6.4f, 6.4f, 6.7e, 6.7f, 6.10e, and 6.10f shows that the forecasting results is near to the actual values. In Figures 6.2, 6.8, 6.5 and 6.11, we superimpose the predictions of all three models and the real price for HSI, N225, DJIA, and S&P 500 respectively. We can observe that ECNN performed better than the other two networks even though there are some regions which show less accuracy for the predicted value.

In Figures 6.3a, 6.3b, 6.3c, 6.3d, 6.3e, and 6.3f we considered loss (MSE) of the HSI forecasting results for 10 and 60 rolling window using the three models mentioned above. Figure 6.3f depicts that ECNN has the least loss values of 3.0×10^{-6} (see Figure 6.3f) as compared to other models (see Figures 6.3b and 6.3d). In addition, there are some points with large mean square errors of the forecasted results of the two models (RNN and LSTM), particularly in the RNN in regions between 100 and 200 mini-batch iterations (see Figure 6.3a), which could be caused by large fluctuations in the HSI price.

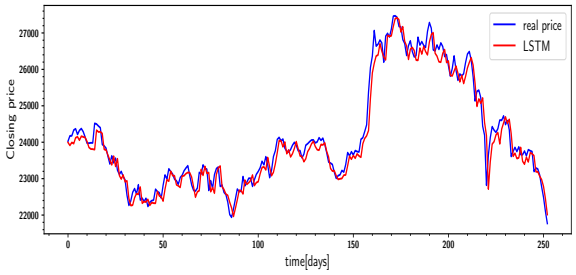
Comparisons for the predicted results for HSI Index



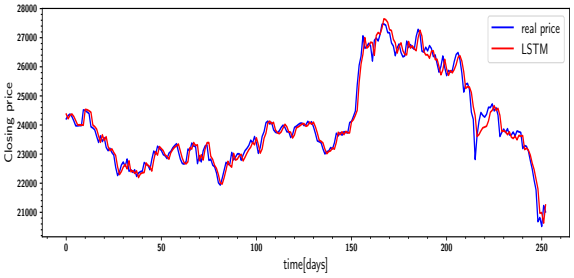
(a) Real and Predicted value of HSI index using RNN, ($\eta = 0.00113$, hidden state size = 32, $rw = 10$).



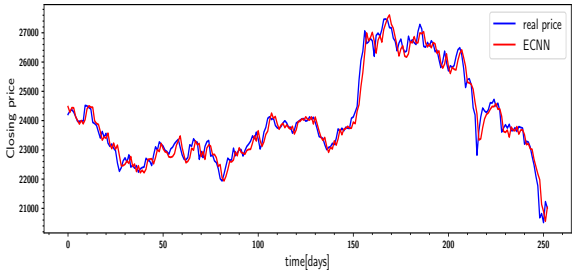
(b) Real and Predicted value of HSI index using RNN, ($\eta = 0.00113$, hidden state size = 32, $rw = 60$).



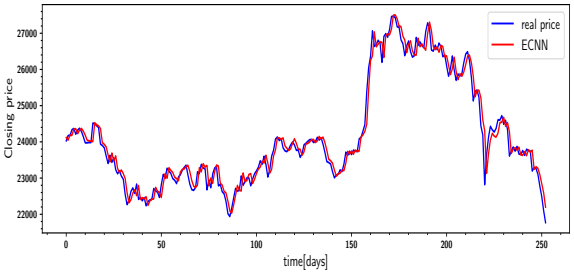
(c) Real and Predicted value of HSI index using LSTM, ($\eta = 0.0011$, hidden state size = 32, $rw = 10$).



(d) Real and Predicted value of HSI index using LSTM, ($\eta = 0.0011$, hidden state size = 64, $rw = 60$).



(e) Real and Predicted value of HSI index using ECNN, ($\eta = 0.00182$, hidden state size = 16, $rw = 10$).



(f) Real and Predicted value of HSI index using ECNN, ($\eta = 0.00182$, hidden state size = 16, $rw = 60$).

Figure 6.1: Comparisons of the predictive data and actual data for the forecasting models. η is the learning rate and rw rolling window. The different hyper-parameters give the best performance of the respective' models.

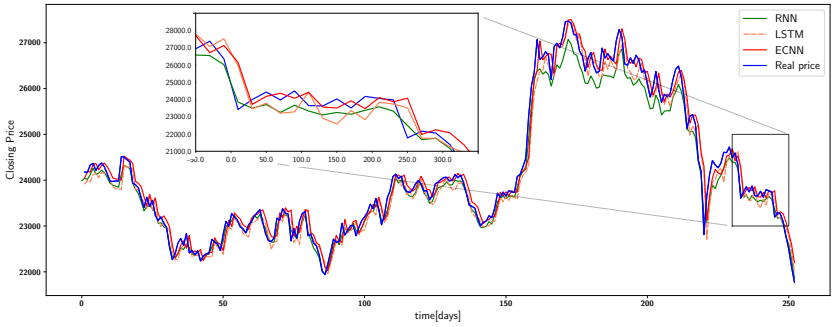
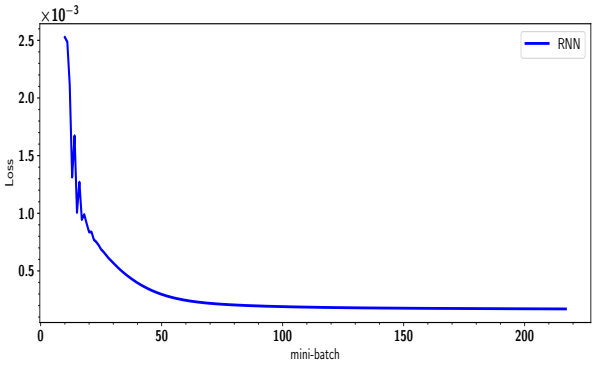
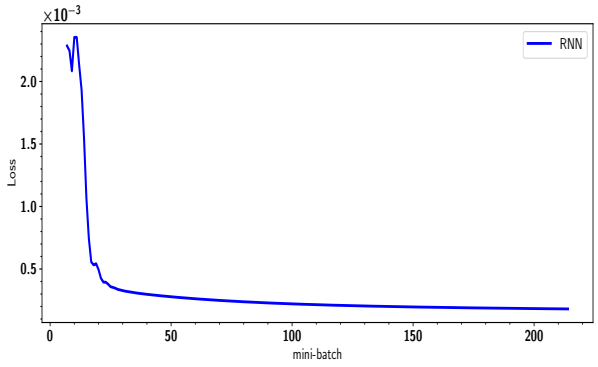


Figure 6.2: Comparisons of actual data and predicted HSI data using RNN, LSTM and ECNN with a window of 60 rolling window.

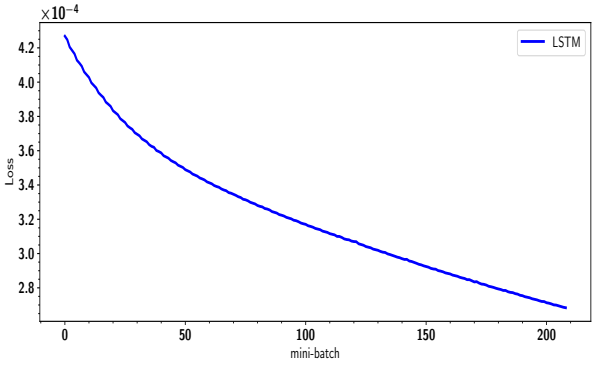
Comparisons for the training loss for HSI Index resulted by the three models



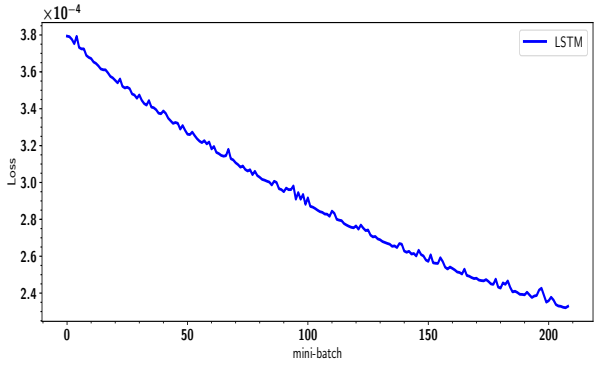
(a) Loss (MSE) obtained over mini-batch training with the RNN model ($rw = 10$).



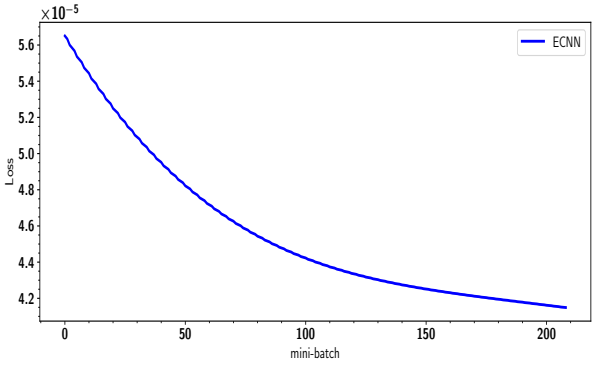
(b) Loss (MSE) obtained over mini-batch training with the RNN model ($rw = 60$).



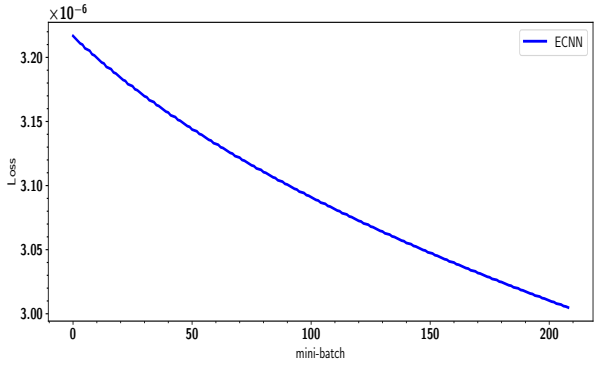
(c) Loss (MSE) obtained over mini-batch training with the LSTM model ($rw = 10$).



(d) Loss (MSE) obtained over mini-batch training with the LSTM model ($rw = 60$).



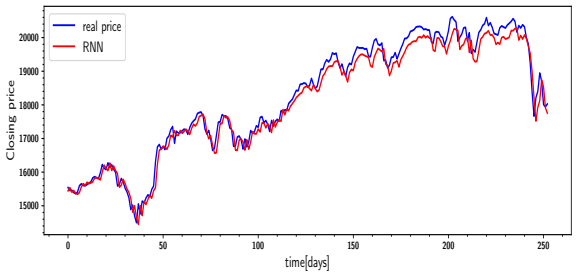
(e) Loss (MSE) obtained over mini-batch training with the ECNN model ($rw = 10$).



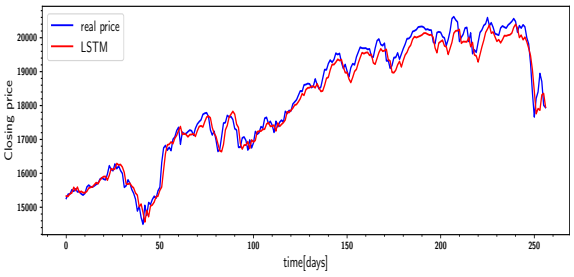
(f) Loss (MSE) obtained over mini-batch training with the ECNN model ($rw = 60$).

Figure 6.3: ((6.3a), (6.3b), (6.3c), (6.3e), (6.3f)) forecasting loss from the models.

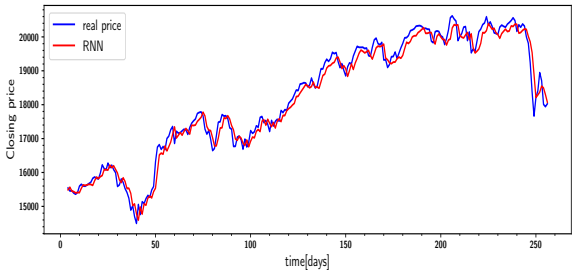
Comparisons of Forecasting Results for N225 Index



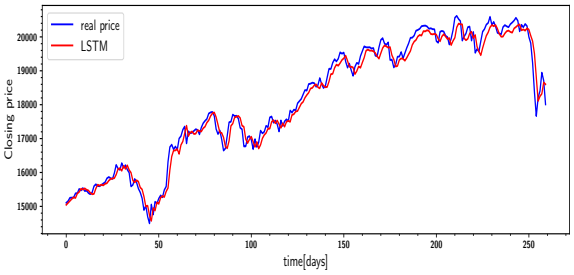
(a) Real and Predicted value of N225 index using RNN, ($\eta = 0.00113$, hidden state size = 32, $rw = 10$).



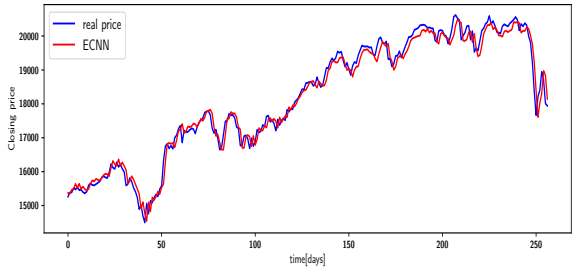
(b) Real and Predicted value of N225 index using RNN, ($\eta = 0.00113$, hidden state size = 32, $rw = 60$).



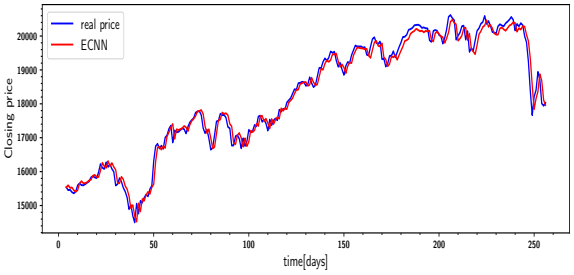
(c) Real and Predicted value of N225 index using LSTM, ($\eta = 0.0011$, hidden state size = 64, $rw = 10$).



(d) Real and Predicted value of N225 index using LSTM, ($\eta = 0.0011$, hidden state size = 64, $rw = 60$).



(e) Real and Predicted value of N225 index using ECNN, ($\eta = 0.00178$, hidden state size = 16, $rw = 10$).



(f) Real and Predicted value of N225 index using ECNN, ($\eta = 0.00178$, hidden state size = 16, $rw = 60$).

Figure 6.4: Comparisons of the predictive data and actual data for the forecasting models. η is the learning rate and rw rolling window. The different hyper-parameters give the best performance of the respective' models.

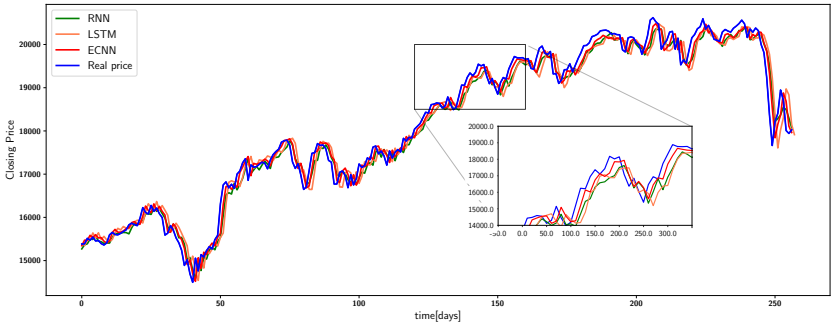
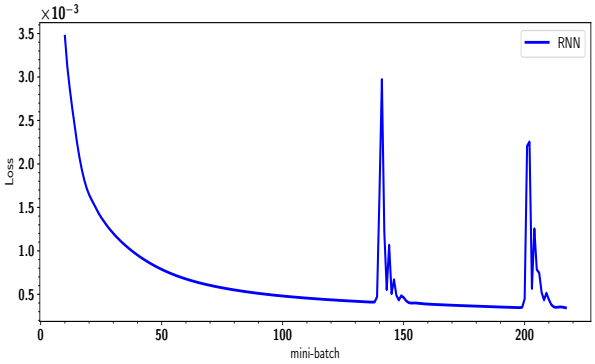
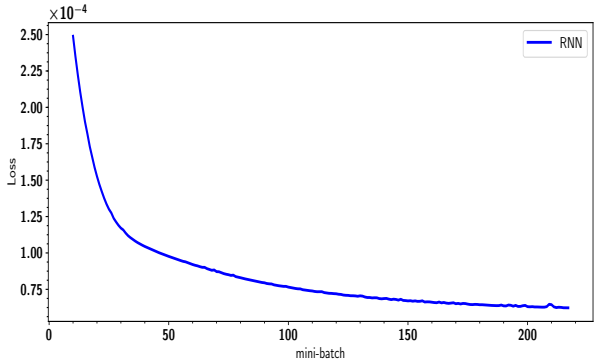


Figure 6.5: Comparisons of actual data and predicted N225 data using RNN, LSTM and ECNN with a window of 60 rolling window.

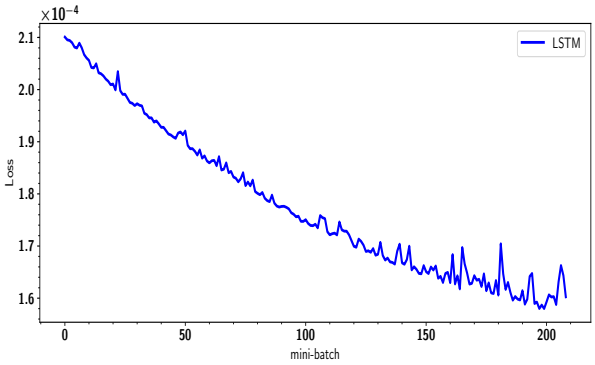
Comparisons for the training loss for N225 Index resulted by the three models



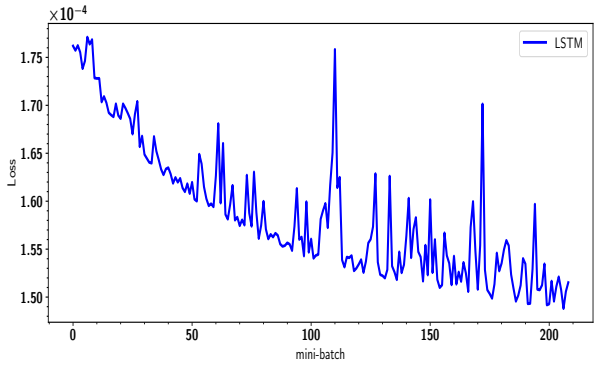
(a) Loss (MSE) obtained over mini-batch training with the RNN model ($rw = 10$).



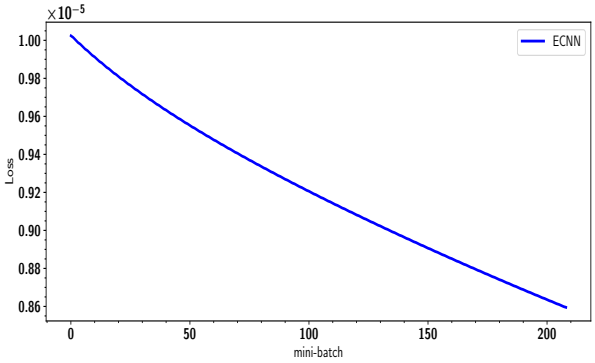
(b) Loss (MSE) obtained over mini-batch training with the RNN model ($rw = 60$).



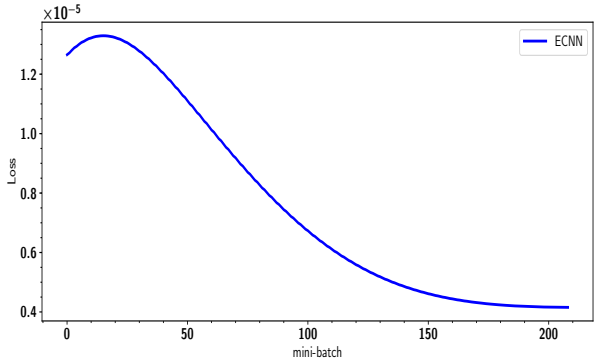
(c) Loss (MSE) obtained over mini-batch training with the LSTM model ($rw = 10$).



(d) Loss (MSE) obtained over mini-batch training with the LSTM model ($rw = 60$).



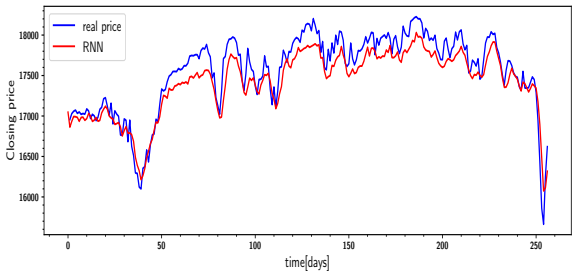
(e) Loss (MSE) obtained over mini-batch training with the ECNN model ($rw = 10$).



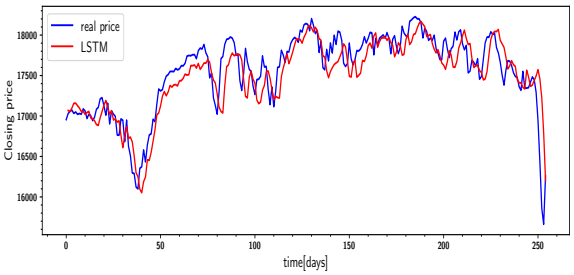
(f) Loss (MSE) obtained over mini-batch training with the ECNN model ($rw = 60$)).

Figure 6.6: ((6.6a), (6.6b), (6.6c), (6.6e), (6.6f)) forecasting loss from the models.

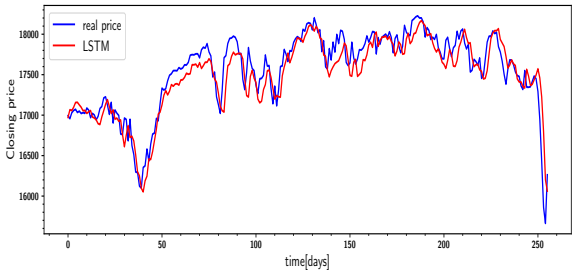
Comparisons of Forecasting Results for DJIA Index



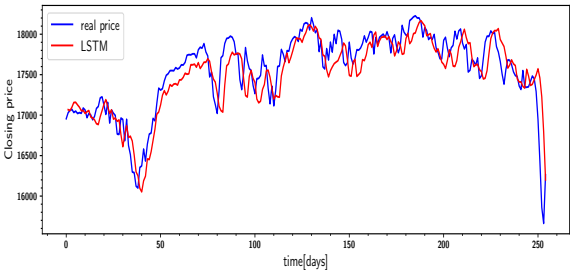
(a) Real and Predicted value of DJIA index using RNN, ($\eta = 0.00121$, hidden state size = 16, $rw = 10$).



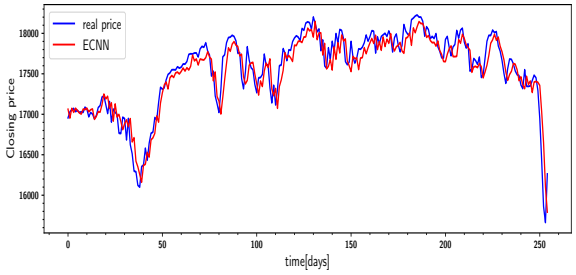
(b) Real and Predicted value of DJIA index using RNN, ($\eta = 0.00121$, hidden state size = 32, $rw = 10$).



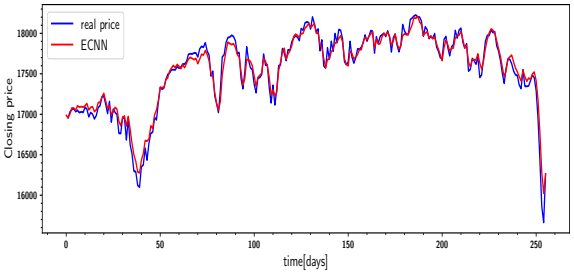
(c) Real and Predicted value of DJIA index using LSTM, ($\eta = 0.00113$, hidden state size = 32, $rw = 10$).



(d) Real and Predicted value of DJIA index using LSTM, ($\eta = 0.00113$, hidden state size = 32, $rw = 10$).



(e) Real and Predicted value of DJIA index using ECNN, ($\eta = 0.00182$, hidden state size = 16, $rw = 10$).



(f) Real and Predicted value of DJIA index using ECNN, ($\eta = 0.00182$, hidden state size = 16, $rw = 10$).

Figure 6.7: Comparisons of the predictive data and actual data for the forecasting models. η is the learning rate and rw rolling window. The different hyper-parameters give the best performance of the respective' models.

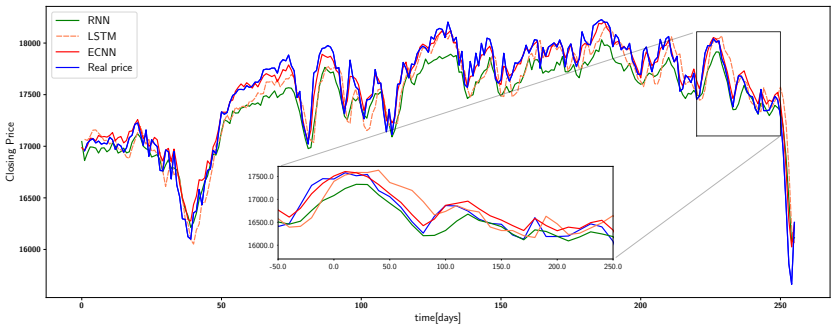
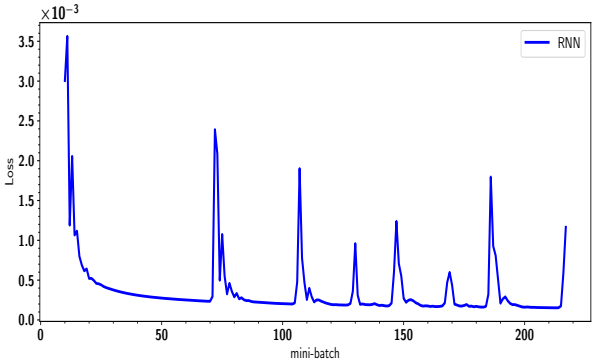
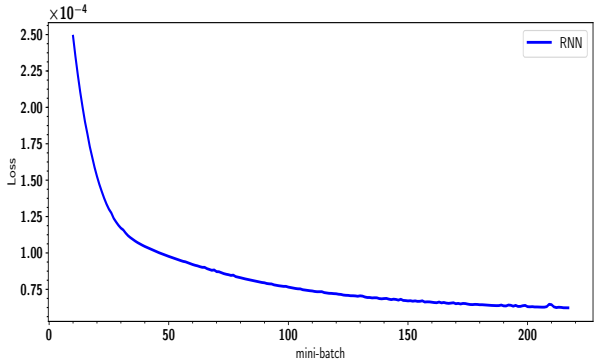


Figure 6.8: Comparisons of actual data and predicted DJIA data using RNN, LSTM and ECNN with a window of 60 rolling window.

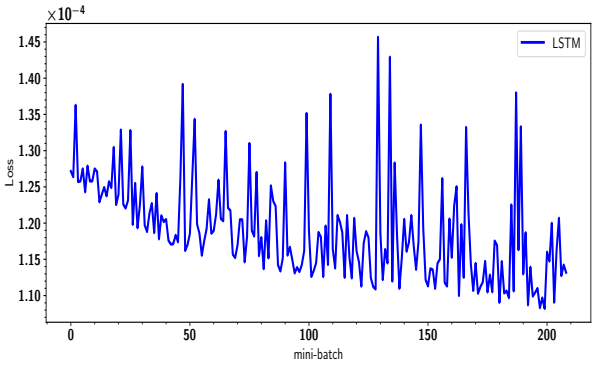
Comparisons for the training loss for DJIA Index resulted by the three models



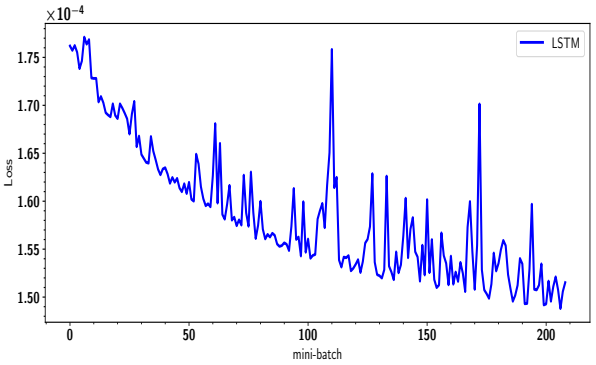
(a) Loss (MSE) obtained over mini-batch training with the RNN model ($rw = 10$).



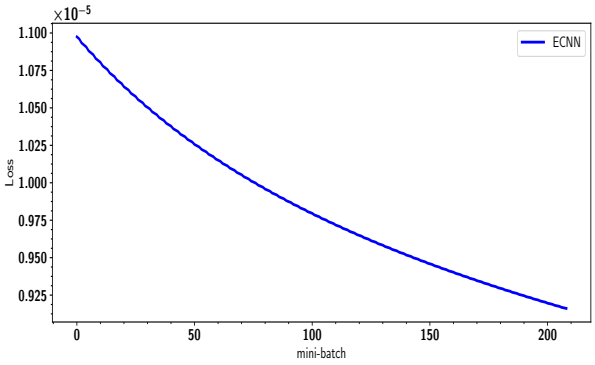
(b) Loss (MSE) obtained over mini-batch training with the RNN model ($rw = 60$).



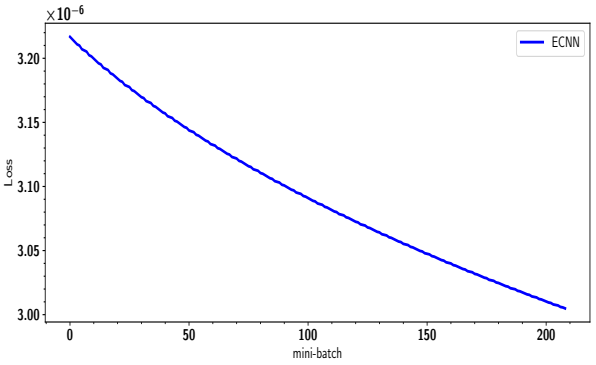
(c) Loss (MSE) obtained over mini-batch training with the LSTM model ($rw = 10$).



(d) Loss (MSE) obtained over mini-batch training with the LSTM model ($rw = 60$).



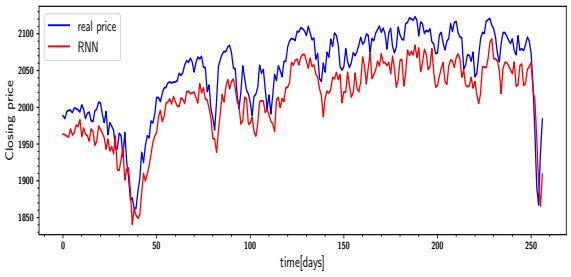
(e) Loss (MSE) obtained over mini-batch training with the ECNN model ($rw = 10$).



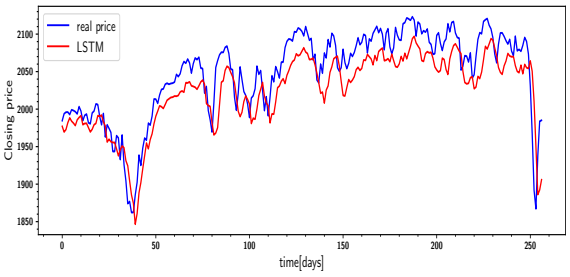
(f) Loss (MSE) obtained over mini-batch training with the ECNN model ($rw = 60$).

Figure 6.9: ((6.9a), (6.9b), (6.9c), (6.9e), (6.9f)) forecasting loss from the models.

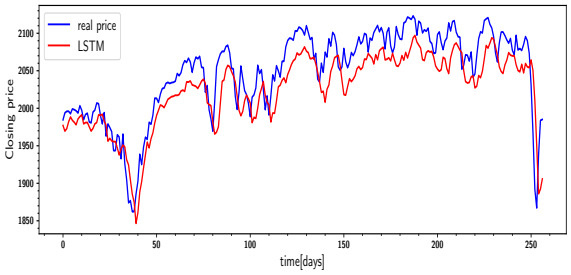
Comparisons of Forecasting Results for S&P 500 Index



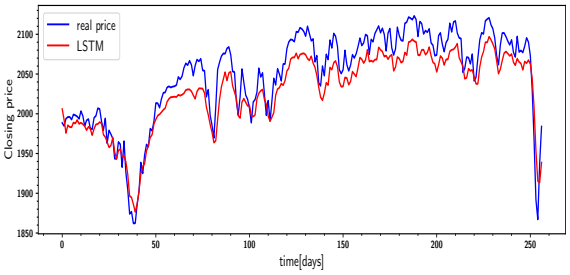
(a) Real and Predicted value of S&P 500 index using RNN, ($\eta = 0.00113$, hidden state size = 32, $rw = 10$).



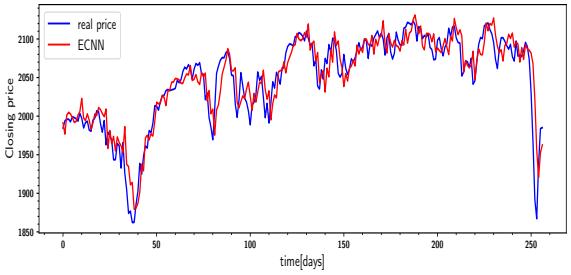
(b) Real and Predicted value of S&P 500 index using RNN, ($\eta = 0.00113$, hidden state size = 32, $rw = 10$).



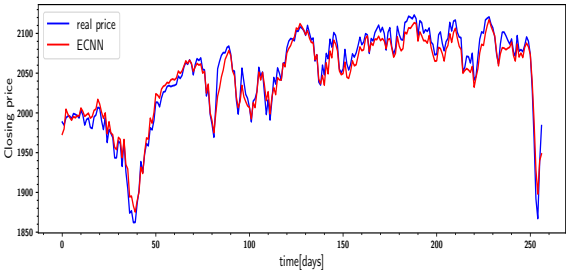
(c) Real and Predicted value of S&P 500 index using LSTM, ($\eta = 0.00113$, hidden state size = 32, $rw = 10$).



(d) Real and Predicted value of S&P 500 index using LSTM, ($\eta = 0.00113$, hidden state size = 32, $rw = 10$).



(e) Real and Predicted value of S&P 500 index using ECNN, ($\eta = 0.00113$, hidden state size = 16, $rw = 10$).



(f) Real and Predicted value of S&P 500 index using ECNN, ($\eta = 0.00113$, hidden state size = 16, $rw = 10$).

Figure 6.10: Comparisons of the predictive data and actual data for the forecasting models. η is the learning rate and rw rolling window. The different hyper-parameters give the best performance of the respective' models.

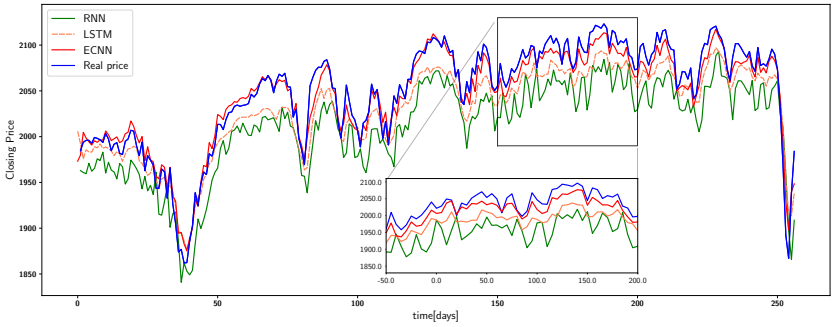
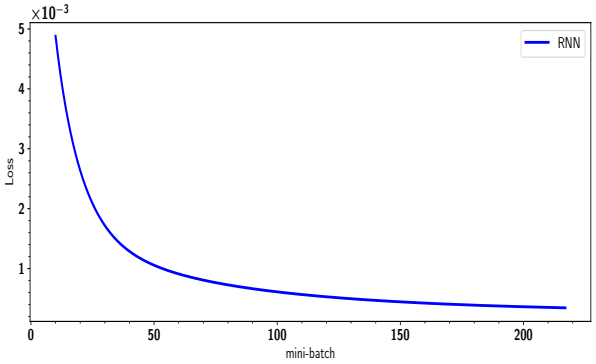
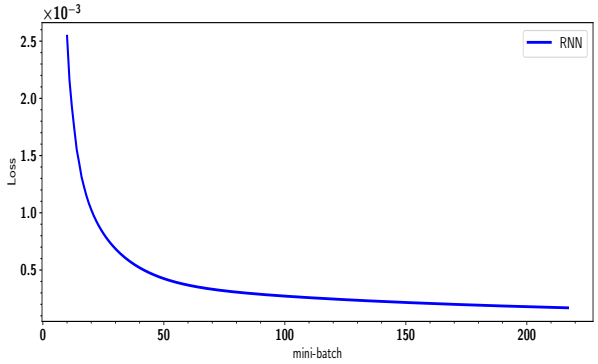


Figure 6.11: Comparisons of actual data and predicted S&P 500 data using RNN, LSTM and ECNN with a window of 60 rolling window.

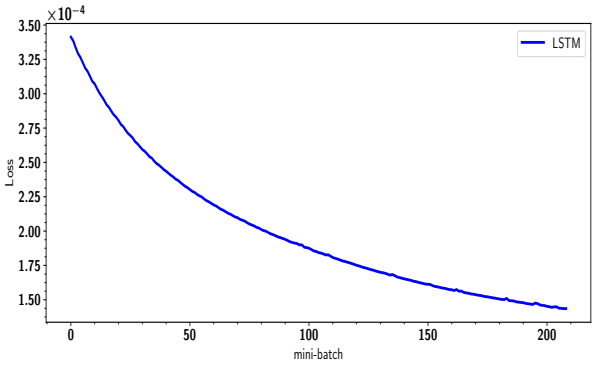
Comparisons for the training loss for S&P 500 Index resulted by the three models



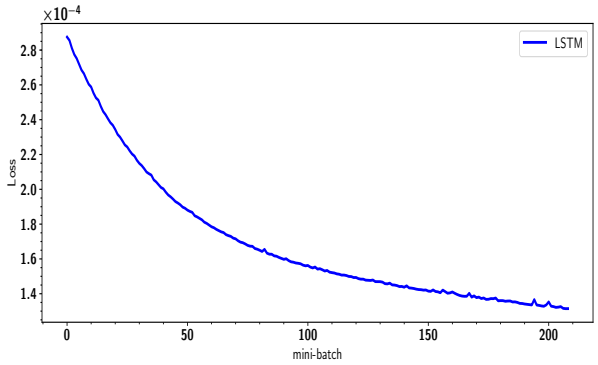
(a) Loss (MSE) obtained over mini-batch training with the RNN model ($rw = 10$).



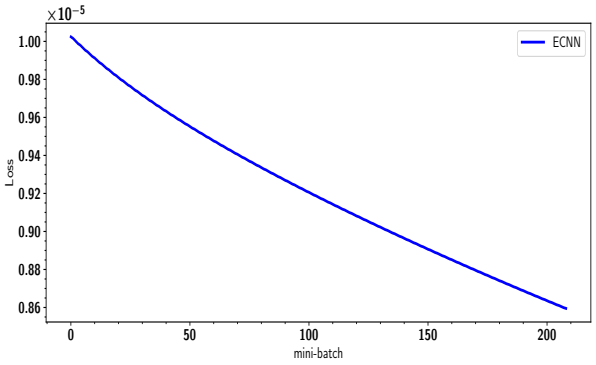
(b) Loss (MSE) obtained over mini-batch training with the RNN model ($rw = 60$).



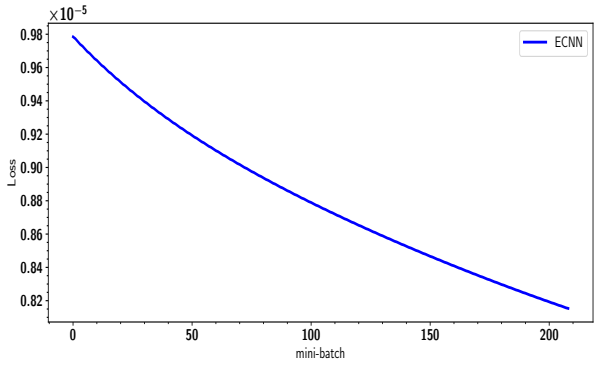
(c) Loss (MSE) obtained over mini-batch training with the LSTM model ($rw = 10$).



(d) Loss (MSE) obtained over mini-batch training with the LSTM model ($rw = 60$).



(e) Loss (MSE) obtained over mini-batch training with the ECNN model ($rw = 10$).



(f) Loss (MSE) obtained over mini-batch training with the ECNN model ($rw = 60$).

Figure 6.12: ((6.12a),(6.12b),(6.12c),(6.12e),(6.12f)) forecasting loss from the models.

6.1 Predictive Accuracy Test

The predictive accuracy test results for each model are reported in Tables 6.1 to 6.24. Each table contains test results in four stock indices trading in similar conditions on the market. In one of our four proposed accuracy indicators, each table shows predictive performance measurement (see Section 5.2.3). We report separately the six years results and the average value over six years for each stock index simultaneously and compare them with the results of Wei, Jun & Yulei results [6]. Tables 6.1, 6.3, 6.5 and 6.7 record model performance in HSI, N225, DJIA and S&P 500 stock indices compared to the results of Wei, Jun & Yulei (see Table 6.2, 6.4, 6.6, and 6.1). From Tables 6.2 to 6.24 it can be seen that ECNN performs much better than the other two models in the prediction of the four stock indices. For example, in predicting HSI index, the average value of MAPE and Theil U of ECNN reach 0.013 and 0.008 (see Table 6.1 and 6.17), respectively, which is much less than those of the other models. Furthermore, the indicator R has an average value of 0.958 (see Table 6.9) for the ECNN model, the highest of the three models. In fact, ECNN not only outperforms the other two models on average but every year.

Based on the results presented in Table 6.3-6.24 of the N225, DJIA and S&P 500, trained networks for all models are valid, as their MAPE values are quite low and acceptable, the lesser MAPE ratio the better. According to the results of forecasting N225, shown in Tables 6.1 and 6.2, ECNN performs better ($0.014 < 0.030 < 0.036$) than LSTM and RNN, respectively. Similarly, for the other stock indices (DJIA and S&P 500), we note that ECNN still outperform the other two models (see Tables 6.5 and 6.7). The MAPE results from the ECNN model also outperform Wei, Jun & Yulei results [6] in all the selected stock indices (see Table 6.1 to 6.8).

Performance of the models, MAPE of evaluation results for the selected data

	Hang Seng index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.031	0.038	0.034	0.040	0.033	0.030	0.034
LSTM	0.027	0.030	0.026	0.021	0.024	0.023	0.025
ECNN	0.013	0.015	0.015	0.011	0.012	0.011	**0.013**

Table 6.1: Performance of the models, MAPE predictive accuracy for HSI.

	Hang Seng Index (Wei, Jun & Yulei results [6])						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.036	0.042	0.030	0.035	0.032	0.034	0.034
LSTM	0.025	0.027	0.024	0.020	0.023	0.023	0.024
WLSTM	0.020	0.027	0.017	0.018	0.028	0.021	0.022
WSAEs-LSTM	0.016	0.017	0.017	0.011	0.021	0.013	**0.015**

Table 6.2: Performance of the models, MAPE predictive accuracy for HSI from Wei, Jun & Yulei results [6].

	Nikkei 225 Index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.043	0.037	0.033	0.035	0.038	0.029	0.036
LSTM	0.037	0.030	0.029	0.031	0.030	0.023	0.030
ECNN	0.012	0.011	0.016	0.011	0.015	0.017	**0.014**

Table 6.3: Performance of the models, MAPE predictive accuracy for N225.

	Nikkei 225 Index (Wei, Jun & Yulei results [6])						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.041	0.039	0.036	0.034	0.034	0.033	0.036
LSTM	0.036	0.030	0.031	0.029	0.028	0.030	0.031
WLSTM	0.033	0.025	0.032	0.025	0.022	0.027	0.028
WSAEs-LSTM	0.020	0.016	0.017	0.014	0.016	0.018	**0.017**

Table 6.4: Performance of the models, MAPE predictive accuracy for N225 from Wei, Jun & Yulei results [6].

	DJIA index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.034	0.038	0.031	0.028	0.031	0.037	0.033
LSTM	0.020	0.028	0.019	0.023	0.017	0.023	0.022
ECNN	0.008	0.010	0.012	0.009	0.011	0.010	**0.010**

Table 6.5: Performance of the models, MAPE predictive accuracy for DJIA.

	DJIA index (Wei, Jun & Yulei results [6])						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.038	0.040	0.029	0.024	0.038	0.026	0.033
LSTM	0.019	0.026	0.016	0.020	0.020	0.022	0.020
WLSTM	0.015	0.018	0.013	0.011	0.017	0.012	0.014
WSAEs-LSTM	0.016	0.013	0.009	0.008	0.008	0.010	**0.011**

Table 6.6: Performance of the models, MAPE predictive accuracy for DJIA from Wei, Jun & Yulei results [6].

	S&P 500 Index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.024	0.015	0.031	0.015	0.031	0.036	0.025
LSTM	0.031	0.023	0.011	0.029	0.033	0.021	0.024
ECNN	0.009	0.013	0.013	0.007	0.015	0.012	**0.012**

Table 6.7: Performance of the models, MAPE predictive accuracy for S&P 500.

	S&P 500 Index (Wei, Jun & Yulei results [6])						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.017	0.019	0.013	0.019	0.020	0.019	0.018
LSTM	0.021	0.018	0.013	0.012	0.017	0.022	0.017
WLSTM	0.015	0.020	0.012	0.010	0.015	0.015	0.015
WSAEs-LSTM	0.012	0.014	0.010	0.008	0.011	0.010	**0.011**

Table 6.8: Performance of the models, MAPE predictive accuracy for S&P 500 from Wei, Jun & Yulei results [6].

Performance of the models, Coefficient Correlation (R) of evaluation results for the selected data

	Hang Seng Index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.782	0.631	0.671	0.812	0.677	0.791	0.727
LSTM	0.863	0.866	0.815	0.910	0.795	0.841	0.848
ECNN	0.951	0.966	0.931	0.953	0.967	0.981	**0.958**

Table 6.9: Performance of the models, R predictive accuracy for HSI.

	Hang Seng Index (Wei, Jun & Yulei results [6])						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.805	0.679	0.684	0.500	0.870	0.840	0.730
LSTM	0.895	0.817	0.727	0.812	0.932	0.901	0.847
WLSTM	0.935	0.810	0.858	0.833	0.900	0.917	0.876
WSAEs-LSTM	0.944	0.924	0.920	0.927	0.904	0.968	**0.931**

Table 6.10: Performance of the models, R predictive accuracy for HSI from Wei, Jun & Yulei results [6].

	Nikkei 225 Index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.631	0.694	0.893	0.779	0.832	0.833	0.780
LSTM	0.773	0.877	0.927	0.883	0.910	0.818	0.865
ECNN	0.899	0.933	0.989	0.991	0.986	0.963	**0.960**

Table 6.11: Performance of the models, R predictive accuracy for N225.

	Nikkei 225 Index (Wei, Jun & Yulei results [6])						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.694	0.625	0.963	0.575	0.893	0.846	0.766
LSTM	0.759	0.759	0.972	0.596	0.918	0.881	0.814
WLSTM	0.748	0.838	0.973	0.786	0.951	0.906	0.867
WSAEs-LSTM	0.895	0.927	0.992	0.885	0.974	0.951	**0.937**

Table 6.12: Performance of the models, R predictive accuracy for N225 from Wei, Jun & Yulei results [6].

	DJIA Index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.681	0.591	0.814	0.681	0.513	0.419	0.607
LSTM	0.831	0.761	0.733	0.891	0.912	0.737	0.811
ECNN	0.932	0.941	0.930	0.981	0.973	0.954	**0.952**

Table 6.13: Performance of the models, R predictive accuracy for DJIA.

	DJIA Index (Wei, Jun & Yulei results [6])						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.684	0.579	0.871	0.669	0.256	0.699	0.627
LSTM	0.860	0.791	0.948	0.751	0.719	0.786	0.809
WLSTM	0.915	0.871	0.963	0.911	0.817	0.927	0.901
WSAEs-LSTM	0.922	0.928	0.984	0.952	0.953	0.952	**0.949**

Table 6.14: Performance of the models, R predictive accuracy for DJIA from Wei, Jun & Yulei results [6].

	S&P 500 Index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.874	0.915	0.673	0.821	0.787	0.735	0.801
LSTM	0.883	0.921	0.891	0.941	0.960	0.811	0.901
ECNN	0.966	0.985	0.972	0.891	0.974	0.981	**0.962**

Table 6.15: Performance of the models, R predictive accuracy for S&P 500.

	S&P 500 Index (Wei, Jun & Yulei results [6])						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.899	0.909	0.966	0.867	0.618	0.822	0.847
LSTM	0.873	0.905	0.968	0.953	0.795	0.755	0.875
WLSTM	0.917	0.886	0.971	0.957	0.772	0.860	0.894
WSAEs-LSTM	0.944	0.944	0.984	0.973	0.880	0.953	**0.946**

Table 6.16: Performance of the models, R predictive accuracy for S&P 500 from Wei, Jun & Yulei results [6].

Performance of the models, Theil U of evaluation results for the selected data

	Hang Seng Index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.023	0.024	0.019	0.022	0.021	0.019	0.021
LSTM	0.014	0.011	0.013	0.016	0.015	0.015	0.014
ECNN	0.009	0.011	0.006	0.006	0.007	0.008	**0.008**

Table 6.17: Performance of the models, Theil U predictive accuracy for HSI.

	Hang Seng Index (Wei, Jun & Yulei results [6])						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.022	0.026	0.018	0.021	0.020	0.020	0.021
LSTM	0.015	0.017	0.016	0.013	0.014	0.015	0.015
WLSTM	0.012	0.017	0.011	0.011	0.021	0.013	0.014
WSAEs-LSTM	0.011	0.010	0.008	0.007	0.018	0.008	**0.011**

Table 6.18: Performance of the models, Theil U predictive accuracy for HSI from Wei, Jun & Yulei results [6].

	Nikkei 225 Index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.028	0.023	0.021	0.020	0.025	0.024	0.022
LSTM	0.019	0.018	0.021	0.023	0.019	0.020	0.032
ECNN	0.011	0.009	0.010	0.008	0.009	0.011	**0.010**

Table 6.19: Performance of the models, Theil U predictive accuracy for N225.

	Nikkei 225 Index (Wei, Jun & Yulei results [6])						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.026	0.025	0.023	0.021	0.021	0.021	0.023
LSTM	0.022	0.022	0.020	0.018	0.018	0.018	0.020
WLSTM	0.021	0.017	0.021	0.015	0.013	0.017	0.018
WSAEs-LSTM	0.013	0.010	0.010	0.009	0.010	0.011	**0.011**

Table 6.20: Performance of the models, Theil U predictive accuracy for N225 from Wei, Jun & Yulei results [6].

	DJIA Index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.031	0.026	0.018	0.015	0.023	0.018	0.022
LSTM	0.012	0.015	0.013	0.013	0.016	0.014	0.014
ECNN	0.007	0.005	0.010	0.004	0.006	0.007	**0.007**

Table 6.21: Performance of the models, Theil U predictive accuracy for DJIA.

	DJIA Index (Wei, Jun & Yulei results [6])						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.025	0.025	0.019	0.016	0.025	0.017	0.021
LSTM	0.013	0.016	0.010	0.014	0.013	0.014	0.013
WLSTM	0.010	0.012	0.008	0.007	0.011	0.008	0.009
WSAEs-LSTM	0.010	0.009	0.006	0.005	0.005	0.006	**0.007**

Table 6.22: Performance of the models, Theil U predictive accuracy for DJIA from Wei, Jun & Yulei results [6].

	S&P 500 index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.012	0.013	0.012	0.008	0.011	0.014	0.012
LSTM	0.007	0.011	0.009	0.010	0.008	0.012	0.010
ECNN	0.006	0.005	0.007	0.008	0.008	0.006	**0.007**

Table 6.23: Performance of the models, Theil U predictive accuracy for S&P 500

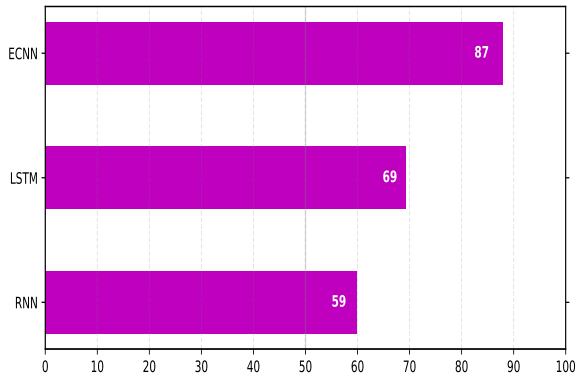
	S&P 500 index (Wei, Jun & Yulei results [6])						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	0.012	0.012	0.009	0.012	0.013	0.012	0.012
LSTM	0.014	0.012	0.009	0.007	0.011	0.016	0.011
WLSTM	0.011	0.014	0.008	0.007	0.011	0.011	0.010
WSAEs-LSTM	0.009	0.010	0.006	0.005	0.008	0.006	**0.007**

Table 6.24: Performance of the models, Theil U predictive accuracy for S&P 500 from Wei, Jun & Yulei results [6].

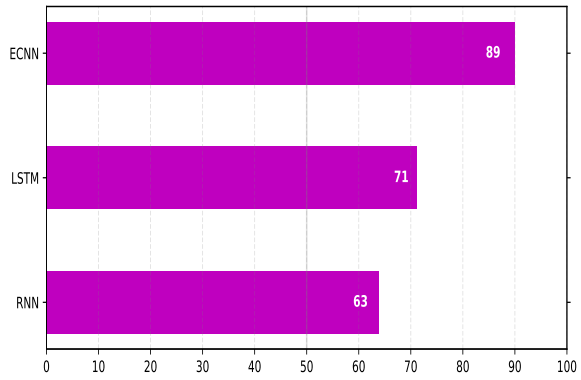
Tables 6.1 to 6.24 represent the models' performance for both relatively developed market (like HSI and N225 index) and developed market (like S&P 500 and DJIA Index). Similar to the discussion we had at the beginning of this section with regards to our findings, we note that ECNN still has the lowest MAPE and Theil U value and the highest R than the other two models, not only from the average value point of view, but also from the annual results point of view. For example, according to the Theil U results of forecasting S&P 500, shown in Table 6.23, ECNN performs better ($0.007 < 0.010 < 0.021$) than LSTM and RNN, respectively. Moreover, ECNN has a higher R value ($0.962 > 0.901 > 0.801$) than LSTM, RNN, respectively. The ECNN has the higher R value in all the selected stock indices compared to WLSTM and WSAEs-LSTM model as shown from Wei, Jun & Yulei results [6] (see Tables 6.9 to 6.16) which indicates better performance for the ECNN model. This concludes that ECNN can achieve stable lower prediction errors and higher predictive accuracy than the other two models, irrespective of market conditions.

6.1.1 Directional Accuracy

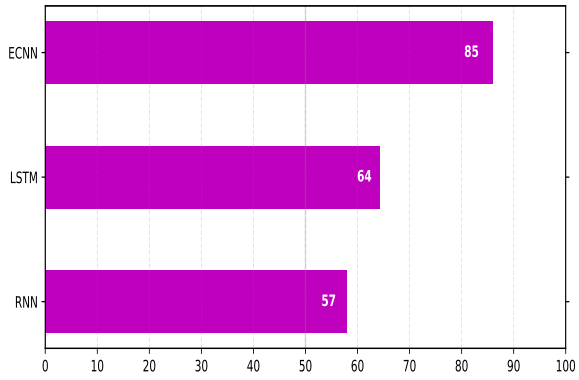
The results of the predictive directional accuracy (DA) (see Section 5.2.3) for each model are shown in Figure 6.13a to 6.13d. Each table report yearly DA over the six years. The ECNN accomplishes the highest DA of 89% for N225 index, which is the highest one among the three models (see Figure 6.13b), while the prediction by RNN is slightly higher than a random guess ($\approx 50\%$). Figure 6.13a, 6.13c, and 6.13d present the models' DA performance in the rest three stock indices. Figure 6.13a, 6.13c demonstrates model DA performance in Hang Sang Index and DJIA Index markets while 6.13d reports the results in S&P 500 Index. Similar to what we found in Figure 6.13b, ECNN still has the highest DA than the other two models. This is the clear indication that ECNN can stably obtain higher directional accuracy than the other two models regardless of market condition.



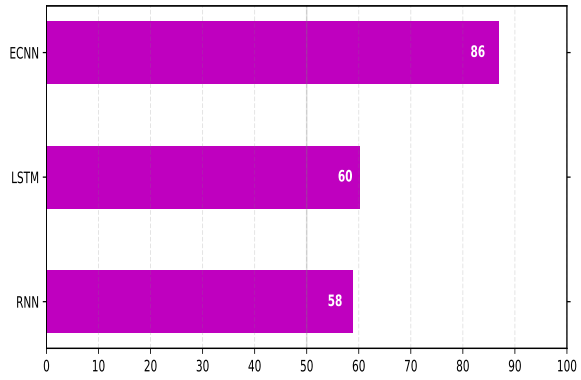
(a) An average yearly directional accuracy (%) of the ECNN, LSTM and RNN for the predicted Hang Sang Index



(b) An average yearly directional accuracy (%) of the ECNN, LSTM and RNN for the predicted Nikkei 225 Index



(c) An average yearly directional accuracy (%) of the ECNN, LSTM and RNN for the predicted DJIA Index



(d) An average yearly directional accuracy (%) of the ECNN, LSTM and RNN for the predicted S&P 500 index Index

Figure 6.13: Comparisons of the predictive directional accuracy a forecasting models in different stock indices.

6.2 Profitability Test

Tables 6.25-6.28 shows the results of the profitability test. We also report annual returns, as well as average six-year returns. Each set of the table described the returns generated by the models for both relatively developed (Hang Sang Index, Nikkei 225) and developed markets (S&P 500 and DJIA) for a specific market condition. In each table, the last row reports returns in the trading of the specific stock index of the buy-and-hold strategy. Tables 6.25 and 6.26 represent the profitable performance for the relatively developed market. The top part is the trading performance based on the predicted Hang Sang Index data, while the bottom part is the trading performance based on the predicted Nikkei 225 Index data. Based on the results of Tables 6.25 and 6.26, the future market of NIKKEI also confirms ECNN 's superiority over LSTM, RNN and buy-and-hold. For example, the average annual earnings of the ECNN model can reach up to 66.901% for Hang Sang and 63.326% for Nikkei 255 Index, while the annual earnings of the other two models are nearly below 30%. Moreover, ECNN model also achieve considerably high performance (see Section 6.1) and return on a yearly bases (see Tables 6.25 and 6.26) for both relatively developed markets.

Relatively developed markets

	Hang Sang Index						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	7.354	44.032	-14.327	-17.327	38.336	41.053	16.520
LSTM	25.565	22.764	3.853	15.143	63.304	27.436	26.344
ECNN	80.862	62.736	77.452	57.537	53.586	69.234	**66.901**
Buy-and-hold	-29.624	19.336	11.311	9.645	-1.775	-2.023	1.145

Table 6.25: Return of the models for the predicted Hang Sang Index.

	Nikkei 225 Index						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	-5.836	-3.983	-14.530	9.524	52.264	25.354	10.465
LSTM	18.083	33.354	-14.835	55.429	7.725	24.926	20.780
ECNN	70.987	41.963	60.332	82.763	49.928	73.983	**63.326**
Buy-and-hold	-18.494	3.163	53.293	11.936	5.442	-8.584	7.792

Table 6.26: Return of the models for the predicted Nikkei 225 Index.

Developed Markets

	S&P 500 index (Our Results)						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	17.435	12.353	15.833	-9.534	-11.363	6.372	5.183
LSTM	-5.933	19.366	26.983	-1.936	3.825	44.272	14.429
ECNN	77.375	52.574	41.523	11.936	60.725	50.353	**49.083**
Buy-and-hold	-11.663	23.385	12.982	14.183	-6.339	7.329	6.646

Table 6.27: Return of the models for the predicted S&P 500 Index.

	DJIA index						
Models	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Year 6**	**Average**
RNN	8.364	3.436	11.117	6.364	-3.448	40.262	11.016
LSTM	44.284	31.442	21.383	-6.348	7.376	41.374	23.252
ECNN	79.363	61.813	49.264	43.264	83.943	81.027	**66.446**
Buy-and-hold	-8.383	-6.375	20.374	10.310	-6.825	10.192	3.2155

Table 6.28: Return of the models for the predicted DJIA Index.

Table 6.27 and 6.28 show the returns on trading in developed markets. Similar to findings in relatively developed markets in Table 6.25 and 6.27, ECNN is able to acquire stable earnings each year, while other models experience significant variations in trading earnings. We also note that our proposed model (ECNN) continues to earn the highest profit (see Table 6.26 and 6.27) among the models in our sampled period. Therefore, our findings support that ECNN has the best predictability among the two models.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This study establishes a new forecasting framework to forecast the one-step price closing of four of the world's leading stock market indices (HSI, Nikkei 225, DJIA and S&P 500) traded in different financial markets. The study, has investigated a predictive model known as Error Correction Neural Network proposed by Zimmermann [71] to forecast the mentioned above indices which are then compared with two popular deep neural network models (RNN and LSTM). The procedural development for this framework is as follows:(1)data preprocessing of daily open, low, high and close prices and technical indicators (see Section 5.1.2), which then are used as input features on our models; (2) daily stock market prediction are made by each model, choosing the best hyper-parameter. These predictions are performed for the selected stock indices; (3) based on the predictions, we perform profitability test by generating a six-yearly annual returns for each model on the select stock indices to achieve more empirical outcomes.

The proposed prediction model, ECNN, is compared with existing methods such as the Buy and Hold trading strategy [3], the RNN and the LSTM system. To show that ECNN is robust enough, this trading system has been applied for six years (2010–2016) to four stock markets, namely HSI, N225, DJIA and S&P 500 Index. In comparison with the other two models, we test the predictive accuracy and profitability of our proposed model (ECNN). The results show that both predictive accuracy and profitability can outperform the other two regardless of the stock index selected for examination. In addition, the average ECNN returns are substantially higher than those of previous studies such as [6, 33].

While the proposed model (ECNN) has satisfactory predictive performance, there are still some insufficiencies. For example, a more advanced hyperparameter selection scheme could be added to the system to further optimize the proposed deep learning framework and develop a trading strategy combined with reinforcement learning that takes into account the current market situation. Another approach would be to make the network event-driven so that it can respond to structural changes in the financial market.

Another problem with the method to be tackled is how to determine the unfolding and overflowing time steps of the forecasting model. Additional work is also suggested in Section 4.5, which could improve the method's performance. There are two suggestions for further work, one of which is the extension of the state space reconstruction to the neural network for error correction. The other suggestion relates to the use of weekly data rather than daily

data in forecasting stock market for the proposed models. In any event, the method provides a solid basis on which the problem of forecasting stocks using neural networks can be built or used.

7.2 Future Work

A clear extension of the work in this thesis is to further investigate the performance of the method developed and how it is generalized in the forecasting of additional stocks. The following sections also describe further work to improve the proposed models.

7.2.1 Input Selection

The problem to be addressed by oneself is how to verify the quantity of input variables to be used, which is the balance between the extra noise of another variable and the corresponding input. One way to simplify this problem is to set up a bottleneck network with the ECNN to analyze the basic components [71]. Therefore, a large number of variables can be used to extract a smaller set of basic components from the bottleneck network. The results of Wei, Jun & Yulei [6], for example, have developed a deep learning framework in which wavelet transform (WT) and stacked auto encoders (SAEs) are combined for stock price forecasting. These main components will then be used as input to the neural network that executes the actual predictions (i.e. the neural network for correction of errors). It could be appropriate to use non-linear principal component analysis in combination with neural networks. This can even be a good addition to the correlation measure, as basic components with low non-linear relationships should be removed. For more information on the analysis of the basic components of neural networks, see e.g. McNelis [52] and Reed et al. [54].

7.2.2 Sentiment Analysis

In addition to the extension discussed in Section 7.2.1, we can further develop the project by exploring the use of text mining, clustering, and machine learning models to develop a system that combines technical, stacked autoencoders [6], and sentiment analysis to determine the movement of a stock. Where the final result of the future project is a system comprised of a novel sentiment analysis used as input for the larger error correction neural network, recurrent neural network and long short-term memory.

List of References

[1] Yaser S Abu-Mostafa and Amir F Atiya. Introduction to financial forecasting. *Applied Intelligence*, 6(3):205–213, 1996.

[2] Afrinvest. Businessday/ Afrinvest 30 index. http://www.businessdayonline.com, 2018. [Online; accessed 21-June-2018].

[3] Franklin Allen and Risto Karjalainen. Using genetic algorithms to find technical trading rules1. *Journal of financial Economics*, 51(2):245–271, 1999.

[4] Erdinc Altay and M Hakan Satman. Stock market forecasting: artificial neural network and linear regression comparison in an emerging market. *Journal of Financial Management & Analysis*, 18(2):18, 2005.

[5] Arash Bahrammirzaee. A comparative survey of artificial intelligence applications in finance: artificial neural networks, expert system and hybrid intelligent systems. *Neural Computing and Applications*, 19(8):1165–1195, 2010.

[6] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.

[7] Russell Beale and Tom Jackson. *Neural Computing-an introduction*. CRC Press, 1990.

[8] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8624–8628. IEEE, 2013.

[9] Yoshua Bengio, Ian J Goodfellow, and Aaron Courville. Deep learning. *Nature*, 521(7553):436–444, 2015.

[10] Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.

[11] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[12] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[13] Istvan SN Berkeley. A revisionist history of connectionism. *Unpublished manuscript*, 1997.

[14] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and contro*, 2015.

[15] Pei-Chann Chang, Yen-Wen Wang, and Wen-Ning Yang. An investigation of the hybrid forecasting models for stock price variation in taiwan. *Journal of the Chinese Institute of Industrial Engineers*, 21(4):358–368, 2004.

[16] DK Chaturvedi, PS Satsangi, and PK Kalra. Effect of different mappings and normalization of neural network models. In *Ninth National Power Systems Conference, Indian institute of Technology, Kanpur*, volume 1, pages 377–386, 1996.

[17] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[18] O Emenike Kalu. Forecasting nigerian stock exchange returns: Evidence from autoregressive integrated moving average (arima) model. 2010.

[19] Robert Engle. Garch 101: The use of ARCH/GARCH models in applied econometrics. *Journal of economic perspectives*, 15(4):157–168, 2001.

[20] Robert F Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the Econometric Society*, pages 987–1007, 1982.

[21] Laurene V Fausett et al. *Fundamentals of neural networks: architectures, algorithms, and applications*, volume 3. Prentice-Hall Englewood Cliffs, 1994.

[22] Paul M Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, 47(6):381, 1954.

[23] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[24] Gene H Golub, Michael Heath, and Grace Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[26] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[27] Ralph Grothmann. *Multi agent market modeling based on neutral networks.* PhD thesis, University of Bremen, Germany, 2003.

[28] Zhiqiang Guo, Huaiqing Wang, Quan Liu, and Jie Yang. A feature fusion based forecasting model for financial time series. *PloS one*, 9(6):e101113, 2014.

[29] Simon Haykin. Neural networks: A comprehensive foundation: Macmillan college publishing company. *New York*, 1994.

[30] Thomas Hellström. *A random walk through the stock market*. PhD thesis, Univ., 1998.

[31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[32] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479, 1997.

[33] Tsung-Jung Hsieh, Hsiao-Fen Hsiao, and Wei-Chang Yeh. Forecasting stock markets using wavelet transforms and recurrent neural networks: An integrated system based on artificial bee colony algorithm. *Applied soft computing*, 11(2):2510–2525, 2011.

[34] Yoshua Bengio Ian Goodfellow and Aaron Courville. *Deep Learning*. Book in preparation for MIT Press, 2016.

[35] International Business Machines Corp . (IBM). *Yahoo!Finance*. http://finance.yahoo.com/q?s=ibm, 2017. [Online; accessed 21-May-2017].

[36] S&P Dow Jones Indices. S&p 500. http://us.spindices.com, 2018. [Online; accessed 03/03/2018].

[37] Investopedia. Ḟundamental Analysis. https://www.investopedia.com/terms/f/fundamentalanalysis.asp, 2017 (accessed September 10, 2017). [Online; accessed September 10, 2017].

[38] Richard G Klein. *The human career: human biological and cultural origins*. University of Chicago Press, 2009.

[39] Ching Lee Koo, Mei Jing Liew, Mohd Saberi Mohamad, Mohamed Salleh, and Abdul Hakim. A review for detecting gene-gene interactions using machine learning methods in genetic epidemiology. *BioMed research international*, 2013, 2013.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[41] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[42] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[43] Richard Lippmann. An introduction to computing with neural nets. *IEEE Assp magazine*, 4(2):4–22, 1987.

[44] Spyros Makridakis. Accuracy measures: theoretical and practical concerns. *International Journal of Forecasting*, 9(4):527–529, 1993.

[45] Sandeep Malik and Ashutosh Kumar Bhatt. Review of various forcasting techniques used for financial forecasting. 2015.

[46] Krzysztof Michalak and Piotr Lipinski. Prediction of high increases in stock prices using neural networks. *Neural Network World*, 15(4):359, 2005.

[47] Marvin Minsky and Seymour Papert. *Perceptrons*: An essay in computational geometry, 1969.

[48] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[49] Jacinta Chan Phooi M'ng and Mohammadali Mehralizadeh. Forecasting East Asian Indices Futures via a Novel Hybrid of Wavelet-PCA Denoising and Artificial Neural Network Models. *PloS one*, 11(6):e0156338, 2016.

[50] Kamban Parasuraman and Amin Elshorbagy. Wavelet networks: an alternative to classical neural networks. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 5, pages 2674–2679. IEEE, 2005.

[51] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[52] D McNELIS PAUL. Neural networks in finance. *Editorial Elsevier*, 2005.

[53] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[54] Russell Reed and Robert J MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999.

[55] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[56] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[57] Ilya Sutskever. Training recurrent neural networks. *University of Toronto, Toronto, Ont., Canada*, 2013.

[58] Timo Teräsvirta. Specification, estimation, and evaluation of smooth transition autoregressive models. *Journal of the American Statistical association*, 89(425):208–218, 1994.

[59] Ruey S Tsay. *Analysis of financial time series*, volume 543. John Wiley & Sons, 2005.

[60] DW Van der Merwe and Andries Petrus Engelbrecht. Data clustering using particle swarm optimization. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 1, pages 215–220. IEEE, 2003.

[61] Value Walk. *Stock Splitting And The Market Crash 1929*, 2018 (accessed June 21, 2018).

[62] William WS Wei. Time series analysis. In *The Oxford Handbook of Quantitative Methods in Psychology: Vol. 2*. 2006.

[63] Paul Werbos. Beyond regression: new fools for prediction and analysis in the behavioral sciences. *PhD thesis, Harvard University*, 1974.

[64] Wikipedia. Efficient-market hypothesis. https://en.wikipedia.org/wiki/Efficient-market_hypothesis, 2018 (accessed September 14, 2018). [Online; accessed September 14, 2018].

[65] Jingtao Yao, Chew Lim Tan, and Hean-Lee Poh. Neural networks for technical analysis: a study on klci. *International journal of theoretical and applied finance*, 2(02):221–241, 1999.

[66] Y-Q Zhang, Somasheker Akkaladevi, G Vachtsevanos, and Tsau Young Lin. Granular neural web agents for stock prediction. *Soft Computing*, 6(5):406–413, 2002.

[67] Hans-Georg Zimmermann, Ralph Grothmann, and Carsten Stolz. Multi-agent market modeling: A survey. *Zeitreihenanalyse in der empirischen Wirtschaftsforschung: Festschrift für Winfried Stier zum 65. Geburtstag*, 197, 2004.

[68] Hans Georg Zimmermann and Ralph Neuneier. Combining state space reconstruction and forecasting by neural networks. In *Datamining und Computational Finance*, pages 259–267. Springer, 2000.

[69] Hans-Georg Zimmermann and Ralph Neuneier. Neural network architectures for the modeling of dynamical systems. *A Field Guide to Dynamical Recurrent Networks*, pages 311–350, 2001.

[70] Hans-Georg Zimmermann, Ralph Neuneier, and Ralph Grothmann. Multi-agent modeling of multiple fx-markets by neural networks. *IEEE Transactions on Neural Networks*, 12(4):735–743, 2001.

[71] Hans-Georg Zimmermann, Ralph Neuneier, and Ralph Grothmann. Modeling dynamical systems by error correction neural networks. In *Modelling and Forecasting Financial Data*, pages 237–263. Springer, 2002.

[72] Hans-Georg Zimmermann, Ralph Neuneier, and Ralph Grothmann. Undershooting: modeling dynamical systems by time grid refinements. In *ESANN*, pages 395–400, 2002.

[73] HG Zimmermann, R Grothmann, AM Schaefer, and Ch Tietz. Identification and forecasting of large dynamical systems by dynamical consistent neural networks. *New Directions in Statistical Signal Processing: From Systems to Brain*, pages 203–242, 2006.

[74] David Zipser and Richard A Andersen. A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature*, 331(6158):679, 1988.